PART TWELVE

Conclusions

36

Major Lessons from This Work

In this book we have presented experimental evidence at many levels of detail for a diverse set of hypotheses. As indicated by the chapter and section headings, the major themes of the MYCIN work have many variations. In this final chapter we will try to summarize the most important results of the work presented. This recapitulation of the lessons learned should not be taken as a substitute for details in the sections themselves. We provide here an abstraction of the details, but hope it also constitutes a useful set of lessons on which others can build. The three main sections of this chapter will

- reiterate the main goals that provide the context for the experimental work;
- discuss the experimental results from each of the major parts of the book; and
- summarize the key questions we have been asked, or have asked ourselves, about the lessons we have learned.

If we were to try to summarize in one word why MYCIN works as well as it does, that word would be *flexibility*. By that we mean that the designers' choices about programming constructs and knowledge structures can be revised with relative ease and that the users' interactions with the system are not limited to a narrow range in a rigid form. While MYCIN was under construction, we tried to keep in mind that the ultimate system would be *used* by many doctors, that the knowledge base would be *modified* by several experts, and that the code itself would be *programmed* by several program-

mers.[1] In hindsight, we now see many areas of inflexibility in MYCIN and EMYCIN. For example, the knowledge acquisition system in EMYCIN requires that the designer of a new system express taxonomic knowledge in a combination of rules and contexts; no facile language is provided for talking about such structures. We lose some expressive power because MYCIN's[2] representation of all knowledge in rules and tables does not separate causal links from heuristics. And MYCIN's control structure forecloses the possibility of tight control over the sequence of rules and procedures that should be invoked together. Thus we are recommending that the principle of flexibility be pushed even farther than we were able to do during the last decade.

Two important ingredients of a flexible system are *simplicity* and *modularity*. We have discussed the simplicity of both the representation and control structure in MYCIN, and the modularity of the knowledge base. While simple structures are sometimes frustrating to work with, they do allow access from many other programs. For example, explanation and knowledge acquisition are greatly facilitated because the rules and backward chaining are syntactically simple (without much additional complication in their actual implementation). The semantics of the rules also appear simple, to users at least, because they have been defined that way by persons in the users' own profession.

The modularity of MYCIN's knowledge representation also contributed to its success. The rules were meant to be individual chunks of knowledge that could be used, understood, or modified independently of other rules. McCarthy, in his paper on the Advice Taker (McCarthy, 1958), set as one requirement of machine intelligence that a program be modifiable by giving it declarative statements about new facts and relations. It should not be necessary to reprogram it. That has been one of the goals of all work on knowledge programming, including our own. MYCIN's rules can be stated to the rule editor as new relations and are immediately incorporated into the definition of the system's behavior.

Modularity includes separation of individual "chunks" of knowledge from one another and from the program that interprets them. But it also implies a structuring of the knowledge that allows indexing from many perspectives. This facilitates editing, explanation, tutoring, and interpreting the individual chunks in ways that simple separation does not. In the

[1]As mentioned, LISP provided a good starting place for the development of a system like MYCIN because its programming constructs need not be fixed in type and size and it allows the building of data structures that are executable as code. At the time of system construction, a designer often needs to postpone making commitments about data structures, data types, sizes of lists, and so forth until experimenting with a running prototype. At the time the knowledge base for an expert system is under construction, similar degrees of flexibility are required to allow the program to improve incrementally. At the time a system is run, it needs flexibility in its I/O handling, for example, to correct mistakes and provide different assistance to different users.

[2]In much of this chapter, what we say about the design of MYCIN carries over to EMYCIN as well.

case of MYCIN's rule-based structure, both the elements of data in a rule's premises and the elements of the rule's conclusion are separated and indexed. However, it is now clear that more structuring of a knowledge base than MYCIN supports will allow indexing chunks of knowledge still further, for example to explain the strategies under which rules are interpreted or to explain the relationships among premise clauses.

36.1 Two Sets of Goals

It must be emphasized that the MYCIN experiments were inherently interdisciplinary, and we were thus guided by two distinct sets of issues: medical goals and artificial intelligence goals. They can be seen as two sides of the same coin. We were trying to build an AI system capable of high-performance problem solving in medicine. Yet each side made its own demands, and we were often forced to allocate resources to satisfy one or the other set of concerns.

On the medical side we wanted to demonstrate the sufficiency of symbolic inference rules in medical problems for which statistical and numerical methods had mostly been used previously. We were also trying to find methods that would allow programs to focus on therapy, as well as on diagnosis. We were explicitly trying to address recognized problems in medical practice and found considerable evidence that physicians frequently err in selecting antimicrobial agents. We were trying to develop a consultation model with which physicians would be comfortable because it mirrored their routine interactions with consultants in practice. And we were trying to develop a system that *could* and *would* be used in hospitals and private practice.

On the AI side, as we have said, the primary motivation was to explore the extent to which rules could be used to achieve expert-level problem solving. In DENDRAL, situation-action rules had been used to encode much of the program's knowledge about mass spectrometry, but considerably more knowledge resided in LISP procedures. In MYCIN, we wanted to use rules exclusively, to see if this could be done in a problem area as complex as medicine. The overriding principle guiding us was the belief that the flexibility of a program was increased by separating medical knowledge from procedures that manipulate and reason with that knowledge. We believed that by making the representation more flexible, it would be easier to build more powerful programs in domains where programs grow by accretion.

The previous chapters reflect this duality of goals. It is important to recognize the tensions this duality introduced in order to understand adequately both the descriptions of the experimental work in this book and the underlying motivations for the individual research efforts.

36.2 Experimental Results

Although we were not always explicitly aware of the hypotheses our work was testing, in retrospect a number of results can be stated as consequences of the experiments performed. The nature of experiments in AI is not well established. Yet, as we said in the preface, an experimental science grows by experimentation and analysis of results. The experiments reported here are not nearly as carefully planned as are, for example, clinical trials in medicine. However, once some uncharted territory has been explored, it is possible to review the path taken and the results achieved.

We have used the phrase "MYCIN-like system" in many places to characterize rule-based expert systems, and we have tried throughout the book to say what these are. In summary, then, let us say what we mean by rule-based systems. They are expert systems whose primary mode of representation is simple conditional sentences; they are extensions of production systems in which the concepts are closer in grain size to concepts used by experts than to psychological concepts. Rule-based systems are deductively not as powerful as logical theorem-proving programs because their only rule of inference is *modus ponens* and their syntax allows only a subset of logically well-formed expressions to be clauses in conditional sentences. Their primary distinction from logic-based systems is that rules define facts in the context of how they will be used, while expressions in logic-based systems are intended to define facts independently of their use.[3] For example, the rule A → B in a rule-based system asserts only that fact A is evidence for fact B.

Rule-based systems are primarily distinguished from frame-based systems by their restricted syntax. The emphasis in a rule is on the inferential relationship between facts (for example, "A is evidence for B" or "A causes B"). In a frame the emphasis is on characterizing concepts by using links of many types (including evidential relations).

Rule-based systems are sometimes characterized as "shallow" reasoning systems in which the rules encode no causal knowledge. While this is largely (but not entirely) true of MYCIN, it is not a necessary feature of rule-based systems. An expert may elucidate the causal mechanisms underlying a set of rules by "decompiling" the rules (see Section 29.3.2 for a discussion of decompiling the knowledge on which the tetracycline rule is based). The difficulties that one encounters with an expanded rule set are knowledge engineering difficulties (construction and maintenance of the knowledge base) and not primarily difficulties of representation or interpretation. However, the causal knowledge thus encoded in an expanded rule set would be usable only in the context of the inference chains in which it fits

[3]This way of making the distinction was pointed out by John McCarthy in a private communication.

and would not be as generally available to all parts of the reasoning system as one might like. A circuit diagram and the theoretical knowledge underneath it, in contrast, can be used in many different ways.

Winston (1977) summarized the main features of MYCIN as follows:

1. MYCIN can help physicians diagnose infections.
2. MYCIN is a backward-chaining deduction system.
3. MYCIN computes certainty factors.
4. MYCIN talks with the consulting physician in English.
5. MYCIN can answer a variety of questions about its knowledge and behavior.
6. MYCIN can assimilate new knowledge interactively.

While this is a reasonable summary of what the program can do, it stops short of analyzing how the main features of MYCIN work or why they do not work better. The analysis presented here is an attempt to answer those questions. Not all of the experiments have positive results. Some of the most interesting results are negative, occasionally counter to our initial beliefs. Some experiments were conceived but never carried out. For example, although it was explicitly our initial intention to implement and test MYCIN on the hospital wards, this experiment was never undertaken. Instead the infectious disease knowledge base was laid to rest in 1978[4] despite studies demonstrating its excellent decision-making performance. This decision reflects the unanticipated lessons regarding clinical implementation (described in Part Eleven) that would not have been realized without the earlier work.

Finally, a word about the organization of this section on results. We have described the lessons mostly from the point of view of what we have learned about building an intelligent program. We were looking for ways to build a high-performance medical reasoning program, and we made many choices in the design of MYCIN to achieve that goal. For the program itself, we had to choose (1) a model of diagnostic reasoning, (2) a representation of knowledge, (3) a control structure for using that knowledge, and (4) a model of how to tolerate and propagate uncertainty. We also had to formulate (5) a methodology for building a knowledge base capable of making good judgments. Our working hypothesis, then, was that the choices we made were sufficient to build a program whose performance was demonstrably good.[5] If we had failed to demonstrate expert-level performance, we would have had reason to believe that one or more of our choices had been wrong. In addition, other aspects of the program were

[4]Much of the MYCIN-inspired work reported in this volume was done after this date, however.

[5]Note that sufficiency is a weak claim. We do not claim that any choice we made is *necessary*, nor do we claim that our choices cannot be improved.

also tested: (6) explanation and tutoring, (7) the user interface, (8) validation, (9) generality, and (10) project organization. The following ten subsections review these ten aspects of the program and the environment in which it was constructed.

36.2.1 The Problem-Solving Model

From the point of view of MYCIN's reasoning, the program is best viewed as an example of the *evidence-gathering paradigm.* This can be seen as a form of search, in which the generator is not constructing complex hypotheses from primitive elements but is looking at items from a predefined list. For diagnosis, MYCIN has the names of 120 organisms. (Twenty-five of the possible causes are explicitly linked to evidence through rules, the rest can be reasoned about through links in tables or links to prior cultures. Properties of all of them must be known, including their sensitivities to each of the drugs.) Logically speaking, MYCIN could run down the list one at a time and test each hypothesis by asking what evidence there is for or against it. This would not produce a pleasing consultation, but it would provide the same diagnoses.

This sort of evidence gathering can be contrasted with heuristic search in which a *generator* of hypotheses defines the search space, as in DENDRAL. It also differs from generate-and-test programs in that hypotheses are not considered (or tested) unless there is evidence pointing to them.

Solutions to problems posed to EMYCIN systems are interpretations of the data. EMYCIN implicitly assumes that there is no unique solution to a problem, but that the evidence will support several plausible conclusions from a fixed list. (This is partly because of the uncertainty in both the data and the rules.) The size of the solution space is thus 2^N where N is the number of single conclusions on the fixed list. In MYCIN there are 120 organism names on the list of possible identities. However, it is unlikely that more than a half-dozen organism identities will have sufficient evidence to warrant covering for them. If we assume that MYCIN will cover for the top six candidate organisms in each case, the number of possible combinations[6] in a solution is more like

$$\binom{120}{6}$$

or about 10^9. Obviously, the method of evidence gathering does not generate all of them.

[6]The number of medically meaningful conclusions is actually much fewer because certain combinations are implausible or nearly impossible.

We have used EMYCIN to build systems in a variety of domains of medicine and engineering. An appropriate application of the evidence-gathering model seems to meet most of the following criteria:

- a classification problem in which data are explained or "covered" by hypotheses from a predefined list;
- a problem that is partly defined by explaining, once, a snapshot of data (as opposed to continuous monitoring problems in which hypotheses are revised frequently as more data are collected);
- a problem of sufficient difficulty that practitioners often turn to textbooks or experts for advice;
- a problem of sufficient difficulty that experts require time for reasoning—their solutions are not instantaneous (but neither do they take dozens of hours);
- a problem of narrow enough scope that a knowledge base can be built and refined in a "reasonable" time (where the resources available and the importance of the problem partly define reasonableness);
- a problem that can be defined in a "closed world," i.e., with a vocabulary that covers the problem description space but is still bounded and "reasonably" small.

Additional characteristics of problems suitable for this kind of solution are listed in Section 36.2.9 on the generality of the EMYCIN framework.

36.2.2 Representation

One of MYCIN's most encouraging lessons for designers of expert systems is the extent to which good performance can be attained with the simple syntax of fact triples and conditional rules. MYCIN's rules are augmented with a context tree around which the dialogue is organized, but other EMYCIN systems (e.g., PUFF) use a degenerate tree of only one kind of object. Also, many rules were encoded in a "shorthand" form (as entries in tables). CF's were added to the simple rule form in MYCIN, but again, other EMYCIN systems (e.g., SACON) perform well with categorical rules (all CF's = 1). For many problems, the simple syntax of fact triples and conditional associations among facts is quite appropriate. In Chapter 3 (Section 3.2) we summarized many additional production system enhancements that were developed for MYCIN.

On the other hand, our experience using EMYCIN to build several expert systems has suggested some negative aspects to using such a simple representation for all the knowledge. The associations that are encoded in rules are elemental and cannot be further examined (except through the symbolic text stored in slots such as JUSTIFICATION or AUTHOR). A reasoning program using only homogeneous rules with no internal distinctions among them thus fails to distinguish among:

Chance associations (e.g., proportionally more left-handed than right-handed persons have been infected by *E. coli* at our institution)

Statistical correlations (e.g., meningococcal meningitis outbreaks are correlated with crowded living conditions)

Heuristics based on experience rather than precise statistical studies (e.g., oral administration of drugs is less reliable in children than are injections)

Causal associations (e.g., streptomycin can cause deafness)

Definitions (e.g., all *E. coli* are gram-negative rods)

Knowledge about structure (e.g., the mouth is connected to the pharynx)

Taxonomic knowledge (e.g., viral meningitis is a kind of infection)

The success of MYCIN, which generally does not distinguish among these types of associations, demonstrates that it is possible to build a high-performance program within a sparse representation of homogeneous rules (augmented with a few other knowledge structures). Nevertheless, limited experience with CENTAUR, WHEEZE, NEOMYCIN, and ONCOCIN leads us to believe that the tasks of building, maintaining, and understanding the knowledge base will be easier if the types of knowledge are separated. This becomes especially pertinent during knowledge acquisition (as described in Part Three) and when teaching the knowledge base to students (Part Eight).

Every formalism limits the kinds of things that can be expressed. From the start we were trying to balance expressive power against simplicity and modularity. As in DENDRAL, in MYCIN we departed from a "pure" production rule representation by allowing complex predicates in the left-hand sides of rules and complex actions in the right-hand sides. All of the inferential knowledge was still kept in rules, however. Every rule was augmented with additional information, using property lists. We used the premise and action properties of rule names for inferential knowledge and used the other properties for bookkeeping, literature references, and the like.[7] Meta-rules can reference the values of any of these slots, to focus attention within the backward-chaining flow of control, thereby making it more sensitive to global context.

Many problems require richer distinctions or finer control than MYCIN-like rules provide. A more general representation, such as frames, allows a system designer to make the description of the world more complex. In frames, for instance, it is easier to express the following:

[7]This is the major distinction between our rules and frames. Inference about inheritance of values is not handled implicitly in MYCIN, as it would be in a frame-based system, but is explicitly dealt with in the action parts of the rules (using the context tree). However, there is considerable similarity in the augmented form of MYCIN's rules and frames, and in their expressive power. Although frames are typically used to represent single concepts, whereas rules represent inferential relationships, the structural similarities between these encoding techniques suggest that frame-based and rule-based representations are not a strict dichotomy.

- Procedural knowledge—sequencing tasks
- Control knowledge—when to invoke knowledge sources
- Knowledge of context—the general context in which elements of the knowledge base are relevant
- Inheritance of properties—automatic transfer of values of some slots from parent concepts to offspring
- Distinctions among types of links—parent and offspring concepts may be linked as
  - class and instance
  - whole and part
  - set and subset

The loss of simplicity in the frame representation, however, may complicate the inference, explanation, and knowledge acquisition routines. For example, inheritance of properties will be handled (and explained) differently depending on the type of link between parent and offspring concepts.

There is a trade-off between simplicity and expressive power. A simpler representation is easier to use but constrains the kinds of things a system builder might want to say. There is also a trade-off between generality and the power of knowledge acquisition tools. An unconstrained representation may have the expressive power of a programming language such as LISP or assembly language, but it can be more difficult to debug. There is considerable overlap among the alternative representation methods, and current work in AI is still experimenting with different ways of making this trade-off.

36.2.3 Control of Inferences

A strong result from the MYCIN experiment is that simple backward chaining (goal-driven reasoning) is adequate for reasoning at the level of an expert. As with DENDRAL, it was somewhat surprising that high performance could be achieved with a simple well-known method. The quality of performance is the same as (and the line of reasoning logically equivalent to) that of data-driven or other control strategies. The main virtues of a goal-driven control strategy are simplicity and ability to focus requests for data. It is simple enough to be explained quickly to an expert writing rules, so that he or she has a sense of how the rules will be used. And it allows explanations of a line of reasoning that are generally easily understood by persons requesting advice.

Internally, backward chaining is also simple. Rules are checked for applicability (i.e., the LHS's are matched against the case data to see if the RHS's should be executed) if and only if the RHS's are relevant to the subgoal under consideration. Relevance is determined by an index created

automatically at the time a rule is created, so rule invocation is highly focused. For example, a new rule A → B will be added to the UPDATEDBY list associated with parameter B; then when subgoal B is under consideration only the rules on this list are tried.

We also needed to focus the dialogue, and we did it by introducing the context tree to guide the subgoal selection.[8] In addition, we needed to overcome some of the sensitivity to the *order* of clauses in a rule dictating the order in which subgoals were pursued and questions were asked. Thus the preview mechanism (Chapter 3) was developed to check all clauses of a rule to see if any are known to be false before chaining backward on the first clause. Once the preview mechanism was implemented, we found we could avoid the appearance of stupidity by introducing antecedent rules in order to make definitional inferences immediately upon receiving some data, for example:

SEX OF PT IS MALE → PREGNANCY OF PT IS NO

Then, regardless of where a clause about pregnancy occurred in a rule's premise, the above antecedent relation would keep the backward-chaining control structure from pursuing earlier clauses needlessly for male patients. Without the antecedent rule, however, nonpregnancy would not be known for males until the pregnancy clause caused backward chaining and the above relation (as a consequent rule) caused the system to check the sex of the patient. Without the preview mechanism, earlier clauses would have been pursued (and unnecessary lines of reasoning possibly generated) before the relevance of the patient's sex was discovered.

The main disadvantage of this control strategy is that users cannot interrupt to steer the line of reasoning by volunteering new information. A user can become frustrated, knowing that the system's present line of reasoning will turn out to be fruitless as a result of data that are going to be requested later. This human-engineering issue is discussed again in Section 36.2.7.

We carried the idea of separating knowledge from inference procedures a step further when we separated control strategies from the rule invocation mechanism. One of the elegant points about this experiment is the use of the same rule formalism to encode strategy rules as we use for the medical rules, with attendant use of the same explanation procedures. In Part Nine we discuss writing meta-rules for controlling inference using the same rule formalism, interpreter, and explanation capabilities. There is sufficient generality in this formalism to support meta-level reasoning, as well as meta-meta-level reasoning and beyond. We needed to add some new predicates to talk *about* rules and rule sets. And we needed one change in the interpreter to check for higher-level rules before executing rules

[8]Recall that the context tree was introduced for two other reasons as well: to allow MYCIN to keep track of multiple instances of the same kind of object, and to allow the program to understand hierarchical relationships among entities.

applicable to a subgoal. We did not experiment enough with meta-rules to determine how much expressive power they offer. However, both CENTAUR and NEOMYCIN give some indication of the control and strategy knowledge we need in medical domains, some of which appears difficult to represent in meta-rules because we lack a rich vocabulary for talking about sequences of tasks. Although meta-rules were designed to prune or reorder the set of rules gathered up by the backward-chaining control routine, their implementation is clean because they reference rules at the next lower level by content and not by name; i.e., they do *not* require specification of an explicit sequence of rules to be invoked in order (e.g., Rule 50 then Rule 71 then Rule 39).

Meta-rules allow separation of types of knowledge in ways that are difficult to capture in medical rules alone. Some diagnostic strategies were initially built into the inference procedure, such as exhaustive invocation of rules—an inherently cautious strategy that is appropriate for this medical context but not for all. Sometimes, though, we wanted MYCIN to be more sensitive to context; the age of the patient, for example, may indicate that some rules can be ignored.[9] Meta-rules work because they can examine the contents of rules at the next lower level and reason about them. This is part of the benefit of the flexibility provided by LISP and the simplicity of the rule syntax.

We have little actual experience with meta-rules in MYCIN, however. Because of the cautious strategy of invoking all relevant rules, we found few opportunities for using them. The one or two meta-rules that made good medical sense could be "compiled out" by moving their contents into the rules themselves. For example, "do rules of type A before those of type B" can be accomplished by manually ordering rules on the UPDATEDBY list or manually ordering clauses in rules. The system overhead of *determining* whether there are any meta-rules to guide rule invocation is a high price to pay if all of the rules will be invoked anyway. So, although their potential power for control was demonstrated, their actual utility is being assessed in subsequent ongoing work such as NEOMYCIN (Clancey, 1983).

36.2.4 Inexact Inference

MYCIN is known partly for its model of inexact inference (the CF model), a one-number calculus for propagating uncertainty through several levels of inference from data to hypotheses. MYCIN's performance shows that, for some problems at least, degrees of evidential support can be captured adequately in a single number,[10] and a one-number calculus can be devised

[9]This was not done with meta-rules, however, because it could easily be handled by the preview mechanism and judicious use of screening clauses.

[10]Although the CF model was originally based on separate concepts of belief and disbelief (as defined for MB and MD in Chapter 11), recall that even then the net belief is reflected in a single number and only one number is associated with each inferential rule.

to propagate uncertainty. The one number we actually use is a combination of disparate factors, most importantly strength of inference and utility considerations. Theoretically, it would have made good sense to keep those separate. Heuristically and pragmatically, we were unable to acquire as many separate numbers as we would have needed for Bayesian probability calculations followed by calculations of expected values (utilities) associated with actions and outcomes.

The CF in a rule measures the *increased* strength of the conclusion. In effect, we asked the medical experts "How much more strongly do you believe the conclusion *h after* you know the premises *e* are true than you did before?" If we were dealing strictly with probabilities, which we are not, then the CF for positive evidential support would be a one-number approximation to

$$\frac{P(h|e) - P(h)}{1 - P(h)}$$

The one-number calculus achieves the goals we sought, although without the precision that many persons desire. The combining of uncertainty depends on relatively small numbers of rules being applicable at any point. Otherwise, many small pieces of evidence ultimately boost the support of every hypothesis to 0.99 and we lose distinctions among strengths of support for hypotheses. The effect of the propagation is a modestly accurate clustering of hypotheses by gross measures of evidential strength (HIGH, MEDIUM, LOW, NONE). But within a cluster the ranking of hypotheses is too dependent on the subjectiveness of the CF's, as well as on the certainty propagation scheme, to be taken precisely.

The focus of a decision-making aid, however, needs to be on recommendations for action. Thus it needs costs and benefits, as well as probabilities, associated with various outcomes. When MYCIN recommends treating for *Streptococcus,* for example, it has combined the likelihood of strep with the risk of failing to treat for it. For this reason we now realize it is perhaps more appropriate to think of CF's as measures of *importance* rather than of probability or strength of belief. That is, they measure the increased importance of acting on the conclusion of a rule in light of new evidence mentioned in the premise. For example, self-referencing rules mention the same parameter in both premise and action parts:

$$A \;\&\; B \;\&\; C \rightarrow A$$

Such a rule is saying, in effect, that if you already have reason to believe A, and if B and C are likely in this case, then increase the importance of A. In principle, we could have separated probabilities from utilities. In practice, that would have required more precision than infectious disease experts were willing or able to supply.

The discontinuity around the 0.2 threshold is not a necessary part of the CF model. It was added to the implementation to keep the backward-chaining control structure from expending effort for very small gain. In a data-driven system the data would all be gathered initially, and the inferences, however weak, could be propagated exhaustively. In a goal-driven system, however, the 0.2 threshold is a heuristic that precludes unnecessary questions. In the rule

$$A \ \& \ B \ \& \ C \rightarrow D$$

if any clause is not "true enough," the subsequent clauses will not be pursued. If clause A, after tracing, has not accumulated evidence over the 0.2 threshold then the system will not bother to ask about clauses B and C. In brief, the threshold was invented for purposes of human engineering since it shortens a consultation and reduces the number of questions asked of the user.

This value of the threshold is arbitrary, of course. It should simply be high enough to prevent the system from wasting its time in an effort to use very small pieces of evidence. With a sick patient, there is a little evidence for almost every disease, so the threshold also helps to avoid covering for almost every possible problem. The threshold has to be low enough, on the other hand, to be sure that important conclusions are considered. Once the 0.2 threshold was chosen, CF's on rules were sometimes set with it in mind. For example, two rules concluding *Streptococcus*, each at the CF = 0.1 level, would not be sufficient alone to include *Streptococcus* in the list of possible causes to consider further.[11]

Because we are not dealing with probabilities, or even with "pure" strength of inference alone, our attempt to give a theoretical justification for CF's was flawed. We based it on probability theory and tried to show that CF's could be related to probabilities in a formal sense. Our desiderata for the CF combining function were based on intuitions involving confirmation, not just probabilities, so it is not surprising, in retrospect, that the justification in terms of formal probability theory is not convincing (see Chapter 12). So the CF model must be viewed as a set of heuristics for combining uncertainty and utility, and not as a calculus for confirmation theory. As we noted in Chapter 13, the Dempster-Shafer theory of evidence offers several potential advantages over CF's. However, simplifying assumptions and approximations will be necessary to make it a computationally tractable approach.

In a deductive system the addition of new facts, as axioms, does not change the validity of theorems already proved. In many interesting problem areas, such as medical diagnosis, however, new knowledge can invalidate old conclusions. This is called nonmonotonic reasoning (McDermott

[11]See the exchange of messages at the end of Chapter 10 for a discussion of how this situation arose in the development of the meningitis knowledge base.

and Doyle, 1980) because new inferences are not always adding new conclusions monotonically to the accumulating knowledge about a problem. In MYCIN, early conclusions are revised as new data are acquired—for example, what looked like an infection of one type on partial evidence looks like another infection after more evidence is accumulated. The problems of nonmonotonicity are mostly avoided, though, because MYCIN gathers evidence for and against many conclusions, using CF's to adjust the strength of evidence of each, and only decides at the end which conclusions to retain. As pointed out in Section 29.4.3, self-referencing rules can change conclusions after all the evidence has been gathered and thus may be considered a form of nonmonotonic reasoning.

Quantification of "Soft" Knowledge

We know that the medical knowledge in MYCIN is not precise, complete, or well codified. Although some of it certainly is mathematical in nature, it is mostly "soft" in the sense that it is judgmental and empirical, and there are strong disagreements among experts about the formulation of what is known. Nevertheless, we needed a way of representing the strength of associations in rules and of calculating the strength with which numerous pieces of evidence support a conclusion. We first looked for a calculus of imprecise concepts that did not involve combining numbers. For example, a few pieces of weakly suggestive evidence would combine into moderately suggestive evidence, and many pieces would be strongly suggestive. But how many? And how do the different qualitative degrees combine? We did not like the idea of discrete categories of strength since it introduces discontinuities in the combinations. So we looked for a continuous function that was not overly sensitive to small changes in degrees.

In working with CF's, we found that quantifying soft knowledge does not require fine levels of precision (Chapter 10). That is why this calculus can be used in a practical domain. With several rules providing evidence for a conclusion, the CF's could be written rather roughly and still give the desired effect. We later showed that, for the MYCIN domain, experts did not have to use more than four or five degrees of evidential strength, even though we provided a continuous scale from 0 to 1.

We discovered two styles of rule composition. The first follows our initial belief that rules can be written independently of one another. The CF's are set by experts based on their accumulated experience of how much more likely or important the conclusion is after the premises are known than it is before they are known. This assumes that CF's do not need to be precisely set because (a) the knowledge itself is not precise and (b) about as many rules will have CF's that are "too high" as will have ones that are "too low" (in some undefinable, absolute sense). The second style of setting CF's is more tightly controlled. Each new empirical association of evidence

Data:
Erroneous
Incomplete

Rules:
Erroneous (or only partly correct)
Incomplete

Conceptual framework (domain-dependent and domain-independent parts):
Incorrect vocabulary of attributes, predicates, and relations
Incorrect inference structure
Incomplete set of concepts
Incomplete logical structure

FIGURE 36-1 Sources of uncertainty in rule-based systems.

with a conclusion, in this view, requires examining rules with similar evidence or similar conclusions to see how strong the association should be, relative to the others. For example, to set the CF on a new rule, A → Z, one would look at other rules such as:

$$X \rightarrow Z \text{ (CF = 0.2)}$$

$$Y \rightarrow Z \text{ (CF = 0.8)}$$

Then, if evidence A is about as strong as Y (0.8) and much stronger than X (0.2), the new CF should be set around the 0.8 level. The exchange of messages at the end of Chapter 10 reflects the controversy that arose in our group over these two styles of CF assignment.

In both cases, the sensitivity analysis mentioned in Chapter 10 convinced us that the rules we were putting into MYCIN were not dependent on precise values of CF's. That realization helped persons writing rules to see that they could be indifferent to the distinction between 0.7 and 0.8, for example, and the system would not break down.

Corrections for Uncertainty

There are many "soft" or ill-structured domains, including medical diagnosis,[12] in which formal algorithmic methods do not exist (Pople, 1982). In diagnostic tasks there are several sources of uncertainty besides the heuristic rules themselves. These are summarized in Figure 36-1.

[12]There are so-called *clinical algorithms* in medicine, but they do not carry the guarantees of correctness that characterize mathematical or computational algorithms. They are decision flow charts in which heuristics have been built into a branching logic so that paramedical personnel can use them to provide good care in many commonly occurring situations.

In an empirical domain, the measurements, observations, and terms used to describe data may be erroneous. Instruments sometimes need recalibrating, or electronic noise in the line can produce spurious readings. Some tests are notoriously unreliable. Similarly, observers sometimes make mistakes in noticing or recording data. Among these mistakes is the failure to describe correctly what one sees. This ranges from checking the wrong box to choosing words poorly. The data are often incomplete as well. Tests with the most diagnostic value and least cost or inconvenience are done first, as a matter of general strategy. At any time, there are always more tests to be done (if only to redo an old one) and always new observations to be made (if only to observe the same variables for a few more hours). But some action must eventually be taken on the best available data, even in the absence of complete information.

With the rules, too, it is impossible to guarantee correctness and completeness (Chapter 8). This is not the fault of the expert supplying the rules; it is inevitable in problem areas in which the knowledge is soft.

Finally, the whole conceptual framework may be missing some critical concepts and may contain constructs that are at the wrong level of detail. Domain-independent parts of the framework that may introduce errors into the problem-solving process include the inference structure and the calculus for combining inexact inferences. The domain-dependent aspects of the problem-solving framework include the vocabulary and the conceptual hierarchies used to relate terms. Some questions of chemistry, for example, require descriptions of molecules in terms of electron densities and cannot be answered with a "ball and stick" vocabulary of molecular structure. Similarly, expert performance in medical domains will sometimes require knowledge of causality or pathophysiologic mechanism, which is not well represented in MYCIN-like rules (see Chapter 29).

The best answer we have found for dealing with uncertainty is redundancy. By that we mean using multiple, overlapping sources of knowledge to reach conclusions, and using the overlaps as checks and balances on the correctness of the contributions made by different knowledge sources. In MYCIN we try to exploit the overlaps in the information contributed by laboratory and clinical data, just as physicians must. For example, a high fever and a high white cell count both provide information about the severity of an infection. On the assumption that the correct data will point more coherently to the correct conclusions than incorrect data will, we expect the erroneous data to have very little effect after all the evidence has been gathered. The *absence* of a few data points will also have little overall effect if other, overlapping evidence has been found. Overlapping inference paths, or redundancy in the rules, also helps correct problems of a few incorrect or missing inferences. With several lines of reasoning leading from data to conclusions, a few can be wrong (and a few can be missing), and the system still ends up with correct conclusions.

We recognize that introducing redundant data and inference rules is at odds with the independence assumptions of the CF model. We did not want the system to fail for want of one or two items of information. When we encounter cases with missing evidence, a redundant reasoning path ensures the robustness of the system. In cases where the overlapping pieces of evidence are all present, however, nothing inside the system prevents it from using the dependent information multiple times. We thus have to correct for this in the rule set itself. The dependencies may be syntactic—for example, use of the same concept in several rules—in which case an intelligent editor can help detect them. Or they may be semantic—for example, use of causally related concepts—in which case physicians writing or reviewing the rules have to catch them.

In the absence of prior knowledge about which data will be available for all cases, we felt we could not insist on a vocabulary of independent concepts for use in MYCIN's rules. Therefore, we had to deal with the pragmatic difficulty of sometimes having too little information and sometimes having overlapping information. Our solution is also pragmatic, and not entirely satisfactory: (a) check for subsumed and overlapping rules during knowledge entry so that they can be separated explicitly; (b) cluster dependent pieces of evidence in single rules as much as possible; (c) organize rules hierarchically so that general information will provide small evidence and more specific information will provide additional confirmation, taking notice of the dependencies involved in using both general and specific evidence; (d) set the CF's on dependent rules (including rules in the hierarchy) to take account of the possibilities of reasoning with redundant paths if all data are *included* and reasoning with a unique path if most data are *missing*.

The problems of an incomplete or inappropriate conceptual scheme are harder to fix. In some cases where we have tried, the EMYCIN framework has appeared to be inappropriate, e.g., a constraint satisfaction problem (MYCIN's therapy algorithm) and problems involving tight procedural control (VM and ONCOCIN). In these instances, we have abandoned this approach to the problem because substantial changes to the conceptual scheme would have required rethinking the definitions of all parts of EMYCIN. The domain-dependent parts are under the control of the experts, though, and can be varied more easily. Not surprisingly, experts with whom we have collaborated seem to prefer working largely within one framework. In MYCIN, for example, there was not a lot of mixing of, say, clinical concepts (such as temperature) and theoretical concepts (such as the effect of fever on cellular metabolism). If the conceptual scheme is inappropriate for the problem, then there is no hope at present for incorporating a smooth correction mechanism. We are always tempted to add more parameters and rules before making radical changes in the whole conceptual framework and approach to the problem, so we will be slow to discover corrections for fundamental limitations.

36.2.5 Knowledge Base Construction and Maintenance

One of the major lessons of this and other work on expert systems is that large knowledge bases must be built incrementally. In many domains, such as medicine, the knowledge is not well codified, so it is to be expected that the first attempts to build a knowledge base will result in approximations. As noted earlier, incremental improvements require flexible knowledge structures that allow easy extensions. This means not only that the syntax should be relatively simple but that the system should allow room for growth. Rapid feedback on the consequences of changes also facilitates improvements. A knowledge base that requires extra compilation steps before it can be tried (especially long ones) cannot grow easily or rapidly.

Knowledge acquisition is now seen as the critical bottleneck in building expert systems. We came to understand through this work that the knowledge-engineering process can be seen as a composite of three stages:

1. knowledge base conceptualization (problem definition and choice of conceptual framework);
2. knowledge base construction (within the conceptual framework); and
3. knowledge base refinement (in response to early performance).

In each stage, the limiting factors are (a) the expressive power of the representation, (b) the extent to which knowledge of the domain is already well structured, (c) the ability of the expert to formulate new knowledge based on past experience, (d) the power of the editing and debugging tools available, and (e) the ability of the knowledge engineer to understand the basic structure and vocabulary of the domain and to use the available tools to encode knowledge and modify the framework.

Our experiments focus largely on the refinement stage.[13] Within this stage, the model that we have found most useful is that of debugging in context; an expert can more easily critique a knowledge base and suggest changes to it in the context of specific cases than in the abstract. Initial formulations of rules are often too general since the conceptualization stage appropriately demands generality. Such overgeneralizations can often best be found and fixed empirically, i.e., by running cases and examining the program's conclusions.

One important limitation of our model is its failure to address the problem of integrating knowledge from different experts. For some extensions to the knowledge base there is little difference between refinement by one expert or many. For extensions in which different experts use different concepts (not just synonyms for the same concept), we have no tools

[13]Some work in progress on the ROGET program (Bennett, 1983) attempts to build an intelligent, interactive tool to aid in conceptualization and construction of EMYCIN systems in new domains.

for reaching a consensus.[14] As suggested in Part Three, the best solution we found for this problem was designating a knowledge base "czar" who was responsible for maintaining coherence and consistency of the knowledge base. The process is facilitated, however, by techniques for comparing new rules with previously acquired knowledge and for performing high-level analyses of large portions of the knowledge base (Chapter 8). We found that this static analysis was insufficient, at least in domains in which nonformal, heuristic reasoning is essential. The best test of strength of a knowledge base appears to be empirical. Nevertheless, a logical analysis can provide important cues to persons debugging or extending a knowledge base, for example, in indicating gaps in logical chains of rules.

There are other models for transferring expertise to a program besides knowledge engineering. The war horse of AI is programming each new performance program using LISP (or another favorite language). This is euphemistically called "custom crafting" or, more recently, "procedural embedding of knowledge." In general, it is slower and the result is usually less flexible than with knowledge engineering, as we learned from DENDRAL.

Another model is based on a direct dialogue between expert and program. This would, if successful, eliminate the need for a knowledge engineer to translate and transform an expert's knowledge. Our attempts to reduce our dependence on knowledge engineers, however, have been largely unsuccessful. Some of the tools built to aid the maintenance of a knowledge base (e.g., the ARL editor; see Chapter 15) have been used by both experts and knowledge engineers. TEIRESIAS (Chapter 9) provides a model by which experts can refine a knowledge base without assistance from a knowledge engineer. For very simple domains such tools can probably suffice for use by experts with little training. As the complexity of a domain grows, however, the amount of time experts can spend seems to shrink. So far, the only way we have found around this dilemma is for knowledge engineers to act as "transducers" to help transform experts' knowledge into usable form.

Other models of knowledge acquisition that we considered leave the expert as well as the knowledge engineer out of the transfer process. Two such models are reading and induction. In the reading model, a program scans the literature looking for facts and rules that ought to be included in the knowledge base. We had considered using the parser described in Chapter 33 to read simplified transcriptions of journal articles. But the difficulties described in that chapter led us to believe that there was as much intellectual effort in transcribing articles for such purposes as in formulating rules directly.[15]

---

[14]We do record the author of each rule with date, justification, and literature citations, but these are not used by the program except as text strings to be printed.

[15]More recent work by others at Stanford explores the use of knowledge-based techniques for inferring new medical knowledge from a large data base of patient information (Blum, 1982).

We did not have the resources to experiment with induction in the MYCIN domain. We kept statistics on rule invocations and found them to be somewhat useful in revealing patterns to the knowledge engineers. For example, rules that are never invoked over a set of test cases may be either covering rare circumstances—in which case they are left unchanged—or failing to match because of errors in the left-hand sides—in which case they are modified. Learning new rules by induction is a difficult task when the performance program chains several rules together to link data to conclusions. In these cases, the so-called credit assignment problem—specifically, the problem of deciding which rules are at fault in case of poor performance—demands considerable expertise. In TEIRESIAS, credit assignment was largely turned over to the expert for this reason.

Since knowledge engineering was our primary mode of knowledge acquisition, we found that some interactive tools for building, editing, and checking the knowledge base gave needed assistance to the system builders. This is sometimes referred to as *knowledge programming*—the construction of complex programs by adding declarative statements of knowledge to an inference framework. The emphasis is on transferring the domain-specific knowledge into a framework and not on building up the framework in the first place from LISP programming constructs. At worst, this is accomplished by an expert using an on-line text editor. This is primitive, but if the expert is comfortable with the syntax and the problem-solving framework, a complex system can still be built more quickly than it could if the expert were forced to write new code, keeping track of array indices and go-to loops. There are many higher levels of assistance possible. Considerable error checking can be done on the syntax, and even more help can be provided by an intelligent assistant that understands some of the semantics of the domain. Knowledge programming, with any level of assistance, is one of the powerful ideas to come out of AI work in the 1970s.

36.2.6 Explanation and Tutoring

When we began this work, there had been little attempt in AI to provide justifications of a program's conclusions because programs were mostly used only by their designers. PARRY (Colby, 1981) had a selective trace that allowed designers to debug the system and casual users to understand its behavior. DENDRAL's Predictor also had a selective trace that could explain the origins of predicted data points, but it was used only for debugging. As part of our goal of making MYCIN acceptable to physicians, we tried from the start to provide windows into the contents of the knowledge base and into the line of reasoning. Our working assumption was that physicians would not ask a computer program for advice if they had to treat the program as an unexaminable source of expertise. They normally ask questions of, or consult, other physicians partly for education to help with future cases and partly for clarification and understanding of

the present case. We believe that initial acceptance of an advice-giving system depends on users being able to understand why it provides the advice that it does (Chapter 34). Moreover, physicians are sensitive to well-established legal guidelines that argue against prescribing drugs without understanding why (or whether) they are appropriate.

The Model

The model of explanation in MYCIN is to "unwind the goal stack" in response to a WHY question. That is, when a user wants to know why an item of information is needed, MYCIN's answer is to show the rule(s) that caused this item to be requested. Answers to successive WHY questions show successively higher rules in the stack. For example, in the reasoning chain

$$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$$

MYCIN chains backward from goal E to the primary element A. A user who wants to know why A is requested will see the rule $A \rightarrow B$. A second WHY question (i.e., "WHY do you want to know B?") will cause MYCIN to show the rule $B \rightarrow C$, and so on. Keeping a simple history list of rule invocations is adequate for producing reasonable explanations of the program's line of reasoning, in part because reasoning is explicitly goal-directed. The goals and subgoals provide an overall rationale for the invocation of rules. The history list captures the context in which information is sought as well as the purpose for which it is sought.

But questions asking why MYCIN requests a particular piece of information provide only a small window on the reasoning process. The complementary HOW questions extend the view somewhat by allowing a user to ask how a fact has already been established or will later be pursued. The same history list provides the means for answering HOW questions during a consultation. For example, a user may be told that item A_2 is needed because B is the current goal and there is a rule of the form

$$A_1 \ \& \ A_2 \ \& \ A_3 \rightarrow B$$

where A_1 is already known (or believed) to be true. Then the user may ask how A_1 is known and will then see the rules that concluded it (or be told that it is primary information entered at the terminal if no rules were used). Similarly, the user may ask how A_3 will be pursued if the condition regarding A_2 is satisfied.

Explanations can be much richer. For example, they can provide insights into the structure of the domain or the strategy behind the line of reasoning. All of these extensions require more sophistication than is embodied in looking up and down a history list. This is a minimal explanation

system. It provides reasons that are only as understandable as the rules are, and some can be rather opaque. Looking up or down the goal stack is not always appropriate, but this is all MYCIN can do. Sometimes, for instance, a user would like a justification for a rule in terms of the underlying theory but cannot get it. Moreover, MYCIN has no model of the user and thus cannot distinguish, say, a student's question from a physician's. These issues were discussed at length in Chapters 20 and 29.

At the end of a consultation, a user may ask questions about MYCIN's conclusions (final or intermediate) and will receive answers much like those given during the consultation. General questions about the knowledge base may also be asked. In order to get MYCIN to answer WHY NOT questions about hypotheses that were rejected or never considered, more reasoning apparatus was needed. Since there is no history of rules that were *not* tried, MYCIN needs to read the rules to see which ones might have been relevant and then to determine why they were not invoked.

Tutoring

We had initially assumed that physicians and students would learn about infectious disease diagnosis and therapy by running MYCIN, especially if they asked why and how. This mode of teaching was too passive, however, to be efficient as a tutorial system, so we began to investigate a more active tutor, GUIDON. The program has two parts: (a) the knowledge base used by MYCIN, and (b) a set of domain-independent tutorial rules and procedures.

We originally assumed that a knowledge base that is sufficient for high-performance problem solving would also be sufficient for tutoring. This assumption turned out to be false, and this negative result spawned revisions in our thinking about the underlying representation of MYCIN's knowledge. We concluded that, for purposes of teaching, and for explanation to novices, the facts and relations known to MYCIN are not well enough grounded in a coherent model of medicine (Chapter 29). MYCIN's knowledge is, in a sense, compiled knowledge. It performs well but is not very comprehensible to students without the concepts that have been left out. For example, a MYCIN rule such as

$$A \rightarrow B$$

may be a compilation of several associations and definitions:

$$A \rightarrow A_1$$

$$A_1 \rightarrow A_2$$

$$A_2 \rightarrow B$$

If A_1 and A_2 are not observable phenomena or quantities routinely measured, the only association that matters for clinical practice is A → B. A student *would* gain some benefit from remembering MYCIN's compiled knowledge, but the absence of an underlying model makes it difficult to remember a scattered collection of rules. Additional knowledge of the structure of the domain, and of problem-solving strategies, provides the "glue" by which the rules are made coherent. Recent work at M.I.T. by Swartout (1983) and Patil et al. (1981) has further emphasized this point.

We also believe that an intelligent tutoring program can be devised such that medical knowledge and pedagogical knowledge are explicitly separated. The art of pedagogy, however, is also poorly codified and evokes at least as much controversy as the art of medicine. GUIDON has directed meaningful dialogues with both the MYCIN and SACON knowledge bases, so its pedagogical knowledge (tutoring rules; see Chapter 26) is not specific to medical education. Some of the knowledge about teaching is procedural because the sequence of actions is often important. Thus the pedagogical knowledge is a mixture of rules and stylized procedures.

36.2.7 The User Interface

Consultation Model

We chose to build MYCIN on the model of a physician-consultant who gives advice to other physicians having questions about patient care. Was it a good choice?

Here the answer is ambiguous. From an AI point of view, the consultation model is a good paradigm for an interactive decision-making tool because it is so clear and simple. The program controls the dialogue, much as a human consultant does, by asking for specific items of data about the problem at hand. Thus the program can understand short English responses to its questions because it knows what answers are reasonable at each point in the dialogue. Moreover, it can ask for as much—and only as much—information as is relevant. Also, the knowledge base can be highly specialized because the context of the consultation can be carefully controlled.

A disadvantage of the consultation model as implemented in MYCIN, however, is that it prevents a user from volunteering pertinent data.[16] Although the approach avoids the need for MYCIN to understand free-text data entry, physicians can find it irritating if they are unable to offer key pieces of information and must wait for the program to ask the right question.[17] In addition, MYCIN asks a lot of questions (around 50 or 60,

[16]Our one attempt to permit volunteered information (Chapter 33) was of limited success, largely because of the complexity of getting a computer to understand free text.

[17]The ability to accept volunteered information is a major feature of the PROSPECTOR model of interaction embodied in KAS (Reboh, 1981).

usually), and the number increases as the knowledge base grows. Few physicians want to type answers to that many questions—in fact, few of them want to type anything. With current technology, then, the consultation model increases the cost of getting advice beyond acceptable limits. Clinicians would rather phone a specialist and discuss a case verbally. Moreover, the consultation model sets up the program as an "expert" and leaves the users in the undesirable position of asking a machine for help. In some professions this may be acceptable, but in medicine it is difficult to sell.

One way to avoid the need for typing so many answers is to tap into on-line patient data bases. Many of MYCIN's questions, for example, could be answered by looking in automated laboratory records or (as PUFF now does) could be gathered directly from medical instruments (Aikins et al., 1983). Another way is to wait for advanced speech understanding and graphical input.

The consultation model assumes a cooperative and knowledgeable user. We attempted to make the system so robust that a user cannot cause an unrecoverable error by mistake. But the designers of any knowledge base still have to anticipate synonyms and strange paths through the rules because we know of no safeguards against malice or ignorance. Some medically impossible values are still not caught by MYCIN.[18] If users are cooperative enough to be careful about the medical correctness of what they type, MYCIN's implementation of the consultation model is robust enough to be helpful.

Other Models of Interaction

DENDRAL does not engage a user in a problem-solving dialogue as MYCIN does. Instead, it accepts a set of constraints (interactively defined) that specify the problem, then it produces a set of solutions. This might be called the *"hired gun" model* of interaction: specify the target, accept the results, and don't ask questions.

Recently we have experimented with a *critiquing model* for the ONCOCIN program, an attempt to respond to some of the limitations of the traditional consultation approach. In the critiquing model, a user states his or her own management plan, or diagnosis, and the program interrupts only if the plan is judged to be significantly inferior to what the program would have recommended (Langlotz and Shortliffe, 1983).

The *monitoring model* of the VM program (Chapter 22) follows much the same interactive strategy as that of ONCOCIN—offering advice only when there is a need. In addition, it periodically updates and prints a summary and interpretation of the patient's condition.

---

[18]For example, John McCarthy (maliciously) told MYCIN that the site of a culture was amniotic fluid—for a male patient—and MYCIN incorrectly accepted it (McCarthy, 1983). Nonmedical users (including one of the authors) have found similar "far-out bugs" as a consequence of sheer ignorance of medicine.

English Understanding

We attempted to design a satisfactory I/O package without programming extensive capabilities for understanding English. One of the pleasant surprises was the extent to which relatively simple sentence parsing and generating techniques can be used. In ELIZA, Weizenbaum (1967) showed that a disarmingly natural conversation can be produced by a program with no knowledge of the subject matter. We wanted to avoid the extensive effort of designing a program for understanding even a subset of unrestricted English. Thus we used roughly the same techniques used in ELIZA and in PARRY (Colby, 1981). Our main concern at the beginning was that the subset of English used by physicians was too broad and varied to be handled by simple techniques. This concern was unfounded. Subsequently, we have come to believe that the more technical the domain, the more stylized the communication. Then keyword and phrase matching are sufficient for understanding responses to questions and for parsing questions asked by users. As long as the program is in control of the dialogue, there is little problem with ambiguity because the types of responses a user can give are determined by the program's questions. Even in a mode in which a user asks questions about any relevant topic (Chapter 18), simple parsing techniques are usually adequate because (a) the range of relevance is rather restricted and (b) terms with ambiguity within this range are few in number and are disambiguated by other terms with unique meanings that serve to fix the context.

We did find, however, that our simple parser was not sufficient for understanding many facts presented at once in a textual description of a patient (Chapter 33). The facts picked out of the text were largely correct, but we missed many. We could successfully restrict the syntax of questions a person can ask without overly restricting the nature of the questions. But we found no general forms for facts that gave us assurance that the program could understand the wide variety of verbs used in case descriptions.

There are several shortcomings in MYCIN'S interface that could antagonize physicians.[19] First, it requires that a user *type*. There is a tantalizing possibility of speech-understanding interfaces that accept sentences in large vocabularies from multiple speakers. But these are not here yet, and certainly were only glimmers on the horizon in 1975. Second, MYCIN requires users to provide information that they know is stored on other computers in the same building. We were prepared to string cables among the computers, but the effort and expense were not justified as long as MYCIN was only a research program. Third, as we have noted, MYCIN does not accept volunteered information. Although we experimented with

[19]The lessons learned regarding the limitations of MYCIN's interface have greatly influenced the design of our recent ONCOCIN system (Chapters 32 and 35). That system's domain was selected largely because it provides a natural mechanism for allowing the physician to volunteer patient information (i.e., the flow sheet), and because data can be entered using a special keypad rather than the full terminal keyboard.

programs to permit this kind of interaction (Chapter 33), the theoretical issues involved prevented robust performance and discouraged us from incorporating the facility on a routine basis. Besides, eventually MYCIN asks all questions that it considers relevant, so, in a logical sense, volunteered information is unnecessary. From the users' point of view, however, MYCIN is often too fully in control of the dialogue. Users would like to be able to steer the line of reasoning and get the program to focus on a few salient facts at the beginning. Fourth, as mentioned above, we believe it is important to provide a window into the line of reasoning and the knowledge base. The window that we provide is narrow, however, and lacks the flexibility and clarity that would let a physician see quickly why MYCIN reasons as it does. Part of the difficulty is that the rules provided as explanations often mix strategy and tactics and thus are difficult to understand in isolation. Our more recent work on explanation has begun to look at issues such as these (Chapter 20).

36.2.8 Validation

There are many dimensions to the question "How good is MYCIN?" We have looked in detail at two: (a) How good is MYCIN's performance? and (b) What features would make such systems acceptable to physicians?

Decision-Making Performance

We experimented with three evaluations of MYCIN, each refined in light of our experience with the previous one, and believe that something much like Turing's test can demonstrate the level of performance of an expert system. In the third evaluation, we asked outside experts to rate the conclusions reached by MYCIN, several Stanford faculty, house staff, and students—on the same set of randomly selected, hard cases. Then, as in Turing's test (Turing, 1950), we looked at the statistics of how the outside experts rated MYCIN's performance relative to that of the Stanford faculty and the others. The conclusion from these studies is that MYCIN recommends therapeutic actions that are as appropriate as those of experts on Stanford's infectious disease faculty—as judged by experts not at Stanford. (More precisely, the outside experts disagreed with MYCIN's recommendation no more often than they disagreed with the recommendations of the Stanford experts.)

Although they are reasonably conclusive, studies such as this are expensive. Considerable research time was consumed in the design and execution of the MYCIN studies, and we required substantial contributed time from Stanford faculty, house staff, and students and from outside experts. Moreover, we learned from the earlier studies that we needed to separate the quality of advice from other factors affecting the utility and

acceptance of the program. Thus the final study provides no information about whether the system would be used in practice, what the cost-benefit trade-offs would be, etc. However, we believe that high performance is a *sine qua non* for an expert system and thus deserves separate evaluation early in a program's evolution (see Chapter 8 of Hayes-Roth et al., 1983).

Acceptability

Unfortunately, we still have not fully defined the circumstances under which physicians will use a computer for help with clinical decision making. Only in the recent ONCOCIN work (Chapters 32 and 35) have we shown that physicians can be motivated to use decision aids in carefully selected and refined environments. In the original MYCIN program we had hoped to provide intelligent assistance to clinicians and to be able to demonstrate that the use of a computer reduced the number (and severity of consequences) of inappropriate prescriptions for antibiotics. Physicians in a teaching hospital, however, may not *need* assistance with this problem to the same extent as others—or, even if they do, they do not *want* it. So we found ourselves designing a program largely for physicians not affiliated with universities, with whom we did not interact daily.

In a survey of physicians' opinions (Chapter 34), we confirmed our impression that explanations are necessary for acceptance. If an assistant is unable to explain its line of reasoning, it will not gain the initial confidence of the clinicians who have to take responsibility for acting on its therapy recommendations. There is an element of legal liability here and an element of professional pride. A physician must understand the alternative possible causes of a problem and the alternative treatments, or else he or she may be legally negligent. Also, professionals will generally believe they are right until given reason to think otherwise. We also found that high performance alone was not sufficient reason for a practicing physician (or engineer or technician) to use a consultation program (Shortliffe, 1982a). We thought that finding a medical problem that is not solved well (and finding documentation of the difficulties) was the right starting place. What we failed to see was that adoption of a new tool is not based solely on demonstrated need coupled with demonstrated high performance of the tool. In retrospect, that was naive. Acceptability is different from high performance (Shortliffe, 1982b).

36.2.9 Generality

One of the most far-reaching sets of experiments in this work involved the generalizability of the MYCIN representation scheme and inference engine. We believed the skeletal program could be used for similar problem-solving tasks in other domains, but no amount of analysis and discussion

could have been as convincing as the working demonstrations of EMYCIN in several different areas of medicine, electronics, tax advising, and software consulting. Making the inference engine domain-independent meant we had to write the rule interpreter so that it manipulates only the symbols named in the rules and makes no semantic transformations except as specified in the knowledge base.

However, there are a number of assumptions about the *type* of problem being solved that are built into EMYCIN. We assume, for instance, that the problem to be solved is one of analyzing a static collection of data (a "snapshot"), weighing all relevant evidence for and against competing hypotheses, and recommending some action. The whole formalism loses strength when it is stretched outside the limits of its design. We see parallels with earlier efforts to build a general problem solver; however, the generality of EMYCIN is intended to be strongly bounded.

There is no mystery to how a system (such as MYCIN) can be generalized (to EMYCIN) so that it is applicable to many problems in other domains: *keep the reasoning processes and the knowledge base separate.* However, some of the limiting characteristics of the data, the reasoning processes, the knowledge base, and the solutions are worth repeating.

The Data

EMYCIN was designed to analyze a static collection of data. The data may be incomplete, interdependent, incorrect ("noisy"), and even inconsistent. A system built in EMYCIN can, if the knowledge base is adequate, resolve ambiguities and cope with uncertainty and imprecision in the data. EMYCIN does assume, however, that there is only one set of data to analyze and that new data will not arrive later from experiments or monitoring. The number of elements of data in the set has been small—roughly 20–100—in the cases analyzed by MYCIN and other EMYCIN systems. But there seems to be no reason why more data cannot be accepted.

Reasoning Processes

EMYCIN is set up to reason backward from a goal to the data required to establish it. It can also do some limited forward reasoning within this context. It thus requests the data it needs when they are not otherwise available.

It is an evidence-gathering system, collecting evidence for and against potentially relevant conclusions. It is not set up to reason in other ways, for example, by generating hypotheses from primitive elements and testing them, by instantiating a template, or by refining a high-level description through successive abstraction levels. It can propagate uncertainty from

the data, through uncertain inference rules, to the conclusions. Backtracking is not supported because the system follows all relevant paths.

Overall, the reasoning is assumed to be analytic and not synthetic. Diagnostic and classification tasks fit well; construction and planning tasks do not. The piece of MYCIN that constructs a therapy plan within constraints, for example, was coded as a few rules that call for evaluating specialized procedures (Chapter 6). It is a complex constraint satisfaction problem, with symbolic expressions of constraints. It was not readily coded in MYCIN-like rules because of the numerous comparison operations (for example, "minimizing").

An interpretation of the data, for instance "the diagnosis of the problem," is the usual goal in EMYCIN systems. In at least one case (SACON; see Chapter 16), however, a solution can have a somewhat more prescriptive flavor. Given a description of a problem, SACON does not solve it directly but rather describes what the user should do to solve it. The prescription of what to do "covers" the data in much the same way as a diagnosis covers the data. Because the evidence-gathering model fit this problem, it was not necessary to treat it as a constraint satisfaction problem.

Knowledge Base

The form of knowledge is assumed primarily to be situation-action rules and fact triples (with CF's). Other knowledge structures, such as tables of facts and specialized procedures, are included as well. Since the knowledge base is indexed and is small relative to the rest of the program, the size of the knowledge base should not be a limiting factor for most problems. MYCIN's knowledge base of 450 rules and about 1000 additional facts (in tables) is the largest with which we have had experience, although ONCOCIN is almost that large and is growing rapidly.

Solutions

As mentioned in the discussion of evidence gathering, the solutions are assumed to be subsets of elements from a predefined list. There are 120 organisms in MYCIN's list of possible causes. In this problem area, the evidence is generally considered insufficient for a precise determination of a unique solution or a strictly ordered list of solutions. Because the evidence is almost certainly incomplete in the first 24–48 hours of a severe infection, both MYCIN and physicians are expected to "cover for" a set of most likely and most risky causes. It is not expected that someone can uniquely identify "the cause" of the problem when the data are suggestive but still leave the problem underdetermined.

36.2.10 Project Organization

Funding

Funding for the research presented here was not easy to find because of the duality of goals mentioned above. Clinically oriented agencies of the government were looking for fully developed programs that could be sent to hospitals, private practices, military bases, or space installations. They saw the initial demonstration with bacteremia as a sign that ward-ready programs could be distributed as soon as knowledge of other infections was added to MYCIN. And they seemed to believe that transcribing sentences from textbooks into rules would produce knowledge bases with clinical expertise. Other funding agencies recognized that research was still required, but we failed to convince them that both medical *and* AI research were essential. We felt that the kinds of techniques we were using could help codify knowledge about infectious diseases and could help define a consensus position on issues about which there are differences of medical opinion. But we also felt that the AI techniques themselves needed analysis and extension before they could be used for wholesale extensions to medical knowledge. More generally, we saw medicine as a difficult real-world domain that is typical of many other domains. Failing to find an agency that would support both lines of activity, we submitted separate proposals for the dual lines. After the initial three years of NIH support for MYCIN, only the AI line was funded by the NSF, ONR, and DARPA (in the efforts that produced EMYCIN, GUIDON, and NEOMYCIN). By 1977 our medical collaborators were in transition for other reasons anyway, so we largely stopped developing the infectious disease knowledge base.[20]

Technology Transfer

When we began, we believed in the "better mousetrap" theory of technology transfer: build a high-performance program that solves an important problem, and the world will transfer the technology. We have learned that several elements of this naive theory are wrong. First, there is a bigger difference between acceptability and performance than we appreciated, as mentioned above. Second, there has to be a convenient mechanism of transfer. MYCIN ran only in Interlisp under the TENEX and TOPS-20

[20]That is not to say, however, that all medical efforts stopped. Shortliffe rejoined the project in 1979 and began defining and implementing ONCOCIN. Clancey needed to reformulate MYCIN's knowledge base in a form more suitable for tutoring (NEOMYCIN) and enlisted the help of Dr. Tim Beckett. Several medical problem areas were investigated and prototype systems were built using EMYCIN. These include pulmonary function testing (PUFF), blood clotting disorders (CLOT), and complications of pregnancy (GRAVIDA). And several masters and doctoral students have continued to use medicine as a test-bed for ideas in AI and decision making, causal reasoning, representation and learning. Several projects undertaken after 1977 are included in the present volume.

operating systems. Since hospital wards and physicians' offices do not have access to the same equipment that computer science laboratories do, we would have had to rewrite this large and complex system in another language to run on smaller machines. We were not motivated to undertake this task. Now, however, smaller, cheaper machines are available that do run Interlisp and other dialects of LISP, so technology transfer is much more feasible than when MYCIN was written.

Stability

We were fortunate with MYCIN in finding stability in (a) the goals of the project, (b) the code, and (c) the system environment.

The group of researchers defining the MYCIN project changed as students graduated, as interests changed, and as career goals took people out of our sphere. Shortliffe, Buchanan, Davis, Scott, Clancey, Fagan, Aikins, and van Melle formed a core group, however, that maintained a certain continuity. Even with a fluid group, we found stability in the overall goal of trying to build an AI system with acknowledged medical expertise. Those who felt this was too narrow a goal moved on quickly, while others found this sharp focus to be an anchor for defining their own research. Another anchor was the code itself. Much of any individual's code is opaque to others, and MYCIN contains its share of "patches" and "hacks." Yet because the persons writing code felt a responsibility to leave pieces of program that could be maintained and modified by others, the programming practices of most of the group were ecologically sound.[21] Finally, the stability of Interlisp, TENEX, and the SUMEX-AIM facility contributed greatly to our ability to build a system incrementally. Without this outside support, MYCIN could not have expanded in an orderly fashion and we would have been forced to undertake massive rewrites just to keep old code running.

36.3 Key Questions and Answers

We realize that a book of this size, describing several experiments that are interrelated in complex and sometimes subtle ways, may leave the reader asking exactly what has been learned by the research and what lessons can be borrowed by others already working in the field or about to enter it. This final chapter has attempted to summarize those lessons, but we feel the need to close with a brief list of frequently asked questions and our

[21]Bill van Melle, Carli Scott, and Randy Davis especially enforced this ethic. In particular, van Melle's system-building tools helped maintain the integrity of a rapidly changing, complex system.

answers to them. The responses are drawn from the work described in earlier chapters but are also colored by our familiarity with other work in AI (particularly research on expert systems). Despite the brevity and simplicity of the questions and answers, we feel that they do summarize the key lessons learned in the MYCIN experiments. For those readers who like to start at the end when deciding whether or not to read a book, we hope that the list will pique their curiosity and motivate them to start reading from the beginning.

- *Is a production rule formalism sufficient for creating programs that can reason at the level of an expert?*

  Yes, although we discovered many limitations and modified the "pure" production rule formalism in several ways in order to produce a program that met our design criteria.

- *Is backward chaining a good model of control for guiding the reasoning and the dialogue in consultation tasks?*

  Yes, particularly when the input data must be entered by the user, although for efficiency and human-engineering reasons it is desirable to augment it with forward chaining and meta-level control as well.

- *Is the evidence-gathering model useful in other domains?*

  Yes, there are many problems in which evidence must be gathered and weighed for a set of possible hypotheses. Infectious disease diagnosis is typical of many problems in having a prestored list of hypotheses that defines the search space. It is not the only useful model for hypothesis formation, however. In other problem areas, hypotheses can be synthesized from smaller elements and then evidence gathered for them in a manner closer to the generate-and-test approach. Or evidence can be gathered during the generation of hypotheses, as in the heuristic search model used in DENDRAL.

- *Is the CF model of inexact reasoning sufficiently precise for expert-level performance?*

  Yes, at least in domains where the evidence weights are used to cluster sets of most likely hypotheses rather than to select the "best" from among them. Some domains demand, and supply, finer precision than the CF model supports, but we felt we lost little information in reasoning with the infectious disease rules using the CF model. We would need to perform additional experiments to determine the breadth of the model's applicability, but we recognize that a calculus of more than one number allows finer distinctions.

- *What is the best way to build a large knowledge base?*

  Knowledge engineering is, for now. Because the problem areas we consider most appropriate for plausible reasoning are those that are not already completely structured (e.g., in sets of equations), constructing a

knowledge base requires defining some new structures. Filling out a knowledge base, then, requires considerable testing and refinement in order to forge a robust and coherent set of plausible rules. Knowledge engineering requires a substantial investment in time for both the knowledge engineer and domain expert, but there are currently no better methods for transferring expertise to expert systems.

- *Were we successful in generalizing the problem-solving framework beyond the domain of infectious diseases?*

  Yes, EMYCIN has been demonstrated in many different problem areas. It has limitations, but its value in system building is more dependent on the structural match of the problem to the task of diagnosis than it is on the specific knowledge structures of the subject area.

- *Can the contents of an EMYCIN knowledge base be effectively used alone for tutoring students and trainees?*

  No, the knowledge base does not contain a rich enough model of the causal mechanisms, support knowledge, or taxonomies of a domain to allow a student to build a coherent picture of how the rules fit together or what the best problem-solving strategies are.

- *Is the consultation model of interaction a good one for a decision-making aid for physicians?*

  For physicians the tradeoff between time and benefit is the key consideration. A lengthy consultation will only be acceptable if there are major advantages for the patient or physician to be gained by using the system. For most applications, therefore, a decision-making aid should be integrated with routine activities rather than called separately for formal consultations. For practitioners in other fields, however, the consultation model may be quite acceptable.

- *Is a simple key word and phrase parser powerful enough for natural language interaction between users and a system in a technical domain?*

  Yes, as long as the user can tolerate a stylized interaction and tries to phrase responses and requests in understandable ways. The approach is probably not sufficient, however, for casual users who seldom use a system and accordingly have no opportunity to learn its linguistic idiosyncrasies.

- *Can we prove the correctness of conclusions from MYCIN?*

  No, because the heuristics carry no guarantees. However, we can demonstrate empirically how well experts judge the correctness of a program's conclusions by using a variant of Turing's test.

- *Why is MYCIN not used routinely and why are the rules not published?*

  Although MYCIN gives good advice and has been a marvelous source of new knowledge about expert systems and their design, computers that run Interlisp are still too expensive, and there are enough deficien-

cies in MYCIN's breadth of knowledge and user interface that it would not be a cost-effective tool for physicians to use on such narrow problem areas as meningitis and bacteremia. We have been asked why we have not published MYCIN's rules about infectious diseases as a service to physicians and medical students, even though the system itself is not available. The long answer is in Chapter 29, but the short answer is that it would not be a service. The rules, as written, do not separate the "key" factors from the context-setting factors, they omit many causal mechanisms that relate key factors with conclusions, and they (together with the rule interpreter) embody a strategy of medical diagnosis that is never explicit. They are not readable as text, nor were they intended to be. They make more sense in the context of *use* than they do in isolation.

- *Why does MYCIN work so well?*

  There are many reasons. First, the task was carefully chosen to increase the likelihood of success: infectious disease therapy selection is a combinatorial problem within a restricted (and relatively small) vocabulary, with time available for several seconds of reasoning, and with available local expertise. Also, there is not just one unique solution to a problem, but a set of acceptable conclusions. Second, the simple, modular knowledge structures we used were designed to be easily understood and changed. Thus the knowledge base could be built incrementally with rapid feedback, i.e., without losing much time to radical changes in underlying data structures and access functions. (In addition, the knowledge structures could be used for multiple interrelated purposes, thereby exploiting and further demonstrating the power and utility of a modular representation scheme.) Third, the research team was dedicated and highly motivated to make MYCIN work. Six doctoral dissertations on MYCIN and related programs resulted from these efforts, with at least as much effort expended by others not working for degrees.