PART NINE

Augmenting the Rules

27

Additional Knowledge Structures

We have so far described MYCIN largely in terms of its knowledge base and inference mechanism, and specifically in terms of rules and a rule interpreter that allow high-performance problem solving. In Chapters 27 through 29 we describe additional knowledge structures that increase the flexibility and transparency of MYCIN's knowledge base. We refer to many of these as *meta-level knowledge.*

When we speak of meta-level knowledge we mean nothing more than knowledge *about* knowledge. In a computer program it needs to be represented and interpreted in order to be useful, but the main idea is that it can be an explicit, and flexible, element of expertise. For example, meta-level knowledge can help in modifying an existing rule and in integrating the modification into the whole rule set because it provides additional information about the existing rules to the editor.

The ideas for using meta-level knowledge in MYCIN grew out of several projects that Randy Davis was working on in the mid-1970s. In the context of knowledge acquisition, we had found that the simple rule editor needed more knowledge about the structure and contents of the rules and about the representations of objects (contexts). In the context of explanation, we found that the predicates (such as SAME) used in rules could be matched to keywords in questions much more easily if the structure of the predicates were known to MYCIN. And, in the context of controlling MYCIN's inferences, we saw that rules *about* MYCIN's rules could provide an element of control. Davis was working on solutions to these problems and saw that the common thread that bound these different parts of the TEIRESIAS system together was meta-level knowledge.

Our first instances of domain-independent meta-level reasoning were (a) the unity path mechanism, by which MYCIN checks for a chain of inferences known to be true with certainty (CF = 1.0) before evaluating other rules, and (b) the preview mechanism, by which MYCIN looked over the clauses of a rule before exhaustively evaluating them to see if the conjunction of premise clauses was already falsified by virtue of any clause

already known to be false (or not "true enough"). In both instances, MYCIN is reasoning *about* its rules before executing them. The important difference between these mechanisms and the meta-knowledge that evolved from work by Davis is that the former are buried in the code of the rule interpreter and thus are not open to examination by other parts of the system, or by the user. After these initial meta-level reasoning techniques were added to the rule interpreter, however, Davis was careful to separate any additional meta-level knowledge structures from the editor, explanation generator, and interpreter, just as we had done with the (object-level) medical knowledge. As a result, the new system (MYCIN plus TEIRESIAS) contains considerably more knowledge about its own knowledge structures than did MYCIN alone. Many of these ideas have subsequently been incorporated into EMYCIN. Chapter 28 provides a summary of the knowledge structures used by TEIRESIAS for knowledge acquisition (see Chapter 9) and control of MYCIN's inferences. This was a line of development that was not anticipated in DENDRAL,[1] and its systematic treatment by Davis in his dissertation was an advance for AI.

Bill Clancey was working on GUIDON at about the same time and was discovering that additional knowledge structures, including meta-level knowledge, were essential for tutoring. TEIRESIAS' knowledge about the form and contents of MYCIN's rules was certainly helpful in constructing GUIDON, but Clancey began focusing more on representing MYCIN's *strategies*. In the course of his research, he also uncovered the importance of two additional kinds of knowledge: knowledge about the *structure* of the domain (and thus about the structure of the rule set), and *support* knowledge that justifies individual rules. Chapter 29 is a careful analysis of these three types of meta-level knowledge that Clancey terms "strategic, structural and support knowledge." This analysis was written in 1981–1982 (and published in 1983) and thus is a recent critique of the structure of MYCIN's knowledge base. We were not unaware of many of the issues raised here, but Clancey provides a coherent framework for thinking about them.

27.1 The Context Tree

In the original (1974) version of MYCIN, several knowledge structures had already been added to the basic rule representation, as discussed in Chapter 5. Most notable among these was the context tree, in which we encoded knowledge about relations among the objects mentioned in rules. The discussion here is taken from the EMYCIN manual (van Melle et al., 1981) and explains this important structure in more detail.

[1]We used the term Meta-DENDRAL to refer to the program that inferred new knowledge for DENDRAL, but we did not have a well-developed concept of knowledge about knowledge.

As described in Chapter 15, an EMYCIN knowledge base is composed of factual knowledge about the domain and production rules that control the consultation interaction and make inferences about a case. Of all the structures the expert must specify for an EMYCIN system, the context tree is perhaps the most important, yet the least discussed. The context tree forms the backbone of the consultant, organizing both the conceptual structure of the knowledge base and the basic flow of the consultation interaction. The tree also indicates the goals for which the consultant will initially attempt to determine values. Since the principles for designing new context trees are poorly understood, this discussion provides examples from various existing EMYCIN systems.

The context tree is composed of at least one, but possibly many, context-types. A context-type corresponds to an actual or conceptual entity in the domain of the consultant, e.g., a patient, an aircraft, or an oil well. Each context-type in the context tree is very much like a record declaration in a traditional programming language. It describes the form of all of its instances created during a case. Thus there are two related but distinct aspects of the context tree mechanism: a static tree of *context-types* and a dynamic tree of *context-instances*. The static tree of types is the structure defined by the expert during system construction and forms the knowledge base "core."

The static tree is used to guide the creation of the dynamic context tree of instances during the consultation. These instances are also organized into a tree that has a form reflecting the structure of the static hierarchy. We distinguish these two structures by referring to them as the static tree and the instance tree. A moderately complex example of each of these types of trees for the SACON system is given in the Figures 27-1 and 27-2. In these and later figures, the links, or relationships, among context-types are labeled to show different uses of the tree.

Each knowledge base has one main, or root, context-type for which there will be a single instance for each consultation. It corresponds to the main subject of the consultation. In MYCIN, for example, the main context-type is PATIENT, and consultation provides advice about disease(s) of the patient. In SACON, the main context-type is STRUCTURE, and a consultation gives advice about performing structural analysis on a structure (such as a bridge or an airplane wing).

Some domains are simple enough that no other context-types are needed. PUFF, for example, needed only attributes of the main context PATIENT. However, other systems, such as MYCIN and SACON, require the ability to discuss multiple objects. In these cases, the context-types are organized into a simple tree structure with the main context at the root. For each context-type that is subordinate to another context-type there is an implicit one-to-many relationship between the instances of each type created during a consultation. Thus, for SACON, there can be many SUBSTRUCTURE instances for the single STRUCTURE instance during a case, and there can be several LOADING instances for each SUBSTRUC-

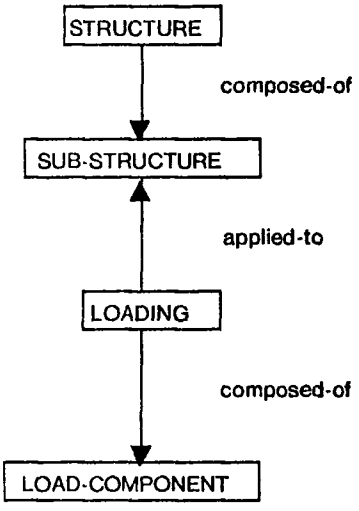

FIGURE 27-1 SACON's static tree of context-types.

TURE instance. It should be noted that, except for the root-type, every possible context-type need not be instantiated during a consultation. In the MYCIN system, for example, the patient may or may not have had any prior drug therapy.

The static tree is the major repository of structural and control information about the consultant. It indicates, in particular, the possible parameters of a context (its PARMGROUP) and the groups of rules that can be applied to instances of a context (its RULETYPES). Hence, the context-types must be defined before one can proceed to acquire rules and parameters, since both of these are defined with respect to the context tree. In addition, the static relationships among the context-types dictate, in large part, the basic mechanism for the propagation of the dynamic tree of instances during a consultation (see Chapter 5).

All of the rules used by the consultant to reason about the domain are written without regard to specific context-instances in an actual consultation. A rule instead refers to parameters of certain context-*types*, and the rule is applied to all the context-instances for which its parameter group is relevant. For example, a rule that concludes about a parameter of a LOADING, say FORCE-BOUND, will be applied to all instances of LOADING, as shown in Figure 27-2 (e.g., LOADING-1, LOADING-2) and may or may not succeed within each instance depending on whether its premise is true in that particular context. In addition, if a rule refers to a specific context-type, its premise can refer to the parameters of any direct ancestors of this context-type. Continuing with our example, the rule premise could refer to parameters of any SUBSTRUCTURE and of the STRUCTURE

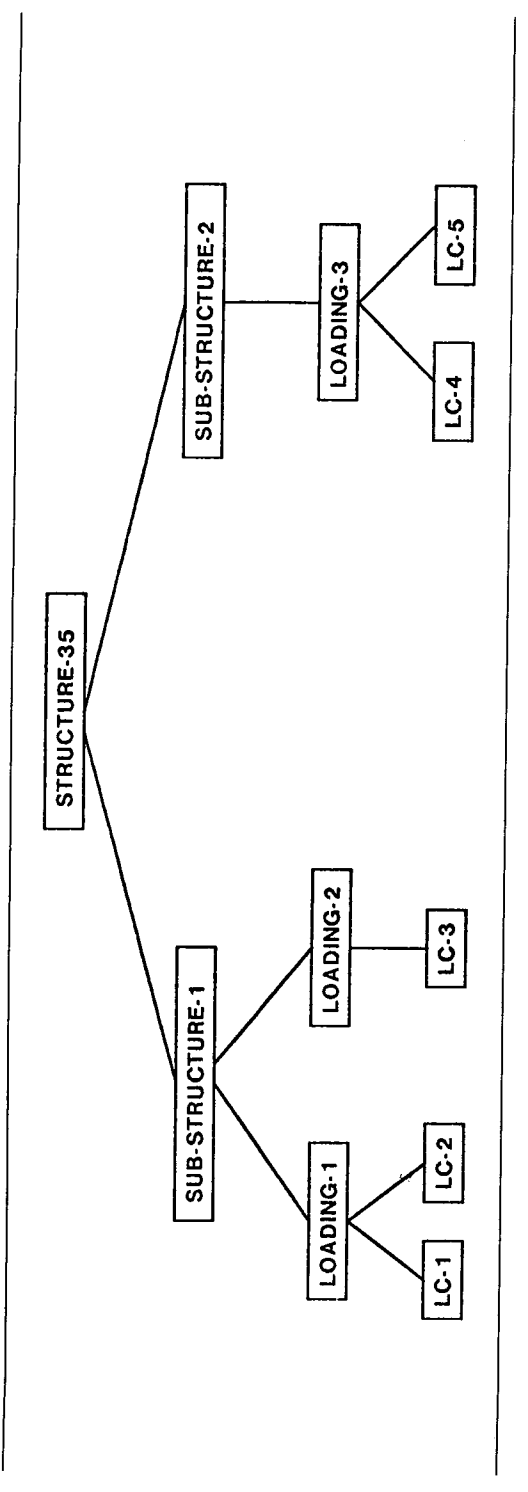

FIGURE 27-2 An instance tree from SACON.

itself. The instance tree organization makes clear which LOADING instances are associated with which SUBSTRUCTURE instance.

If a rule is applied to some context-instance and uses information about context-instances *lower* in the tree, however, an implicit iteration occurs: the rule is applied to each of the lower instances in turn. If the lower context-types have not yet been instantiated, the program digresses to ask about their creation at this time. Thus contexts are instantiated because rules need them,[2] just as parameters are traced when rules need them. In fact, since the goals of the consultation usually consist of finding out something about the root of the tree, the only way that lower context-types are instantiated at all is through the application of rules that use information about lower context-types.

27.1.1 Uses of the Context Tree

There have been a few rather stereotypic uses of the context tree. Although experience to date has by no means exhausted the possible uses, the examples shown here should help readers to understand how an expert and knowledge engineer might select appropriate context-types and organize them in a new domain.

The primary use of additional contexts has been to *structure the data or evidence* to be collected. Thus, in the MYCIN system, the culture contexts describe the tests performed to isolate organisms. Additional information about the patient's current and previous therapies, the cultures, and MYCIN's own estimation of the suspected infections are also represented in the tree. The current context organization for MYCIN is shown in Figure 27-3 and should be contrasted with the sample instance tree of Figure 5-1 (which reflects MYCIN's context-types as they were defined in 1974).[3]

The second major use of the context tree has been to *organize the important components of some object.* For example, in the SACON system the substructures of the main structure correspond to components or regions of the object that have some uniform property, typically a specific geometry or material. Each substructure instance is considered *independently,* and conclusions about individual responses to stress loadings are summarized on the structure level to provide a "global" sense of the overall response of the structure. A recent, additional example of this use of a part-whole hierarchy is found in a system called LITHO (Bonnet, 1979), which interprets data from oil wells. In this system, each well is decomposed into a number of zones that the petrologist can distinguish by depth (Figure 27-4).

A context need not correspond to some physical object but may be an abstract entity. However, the relationships among contexts are explicitly

---

[2]Contexts may also be instantiated by explicit command, but the mechanism is less convenient.

[3]It is instructive to compare this structure with the original context tree described in Chapter 5; the MYCIN system has undergone at least three intermediate reorganizations of its static tree. Significantly, however, the *kinds* of objects in the tree have not changed substantially.

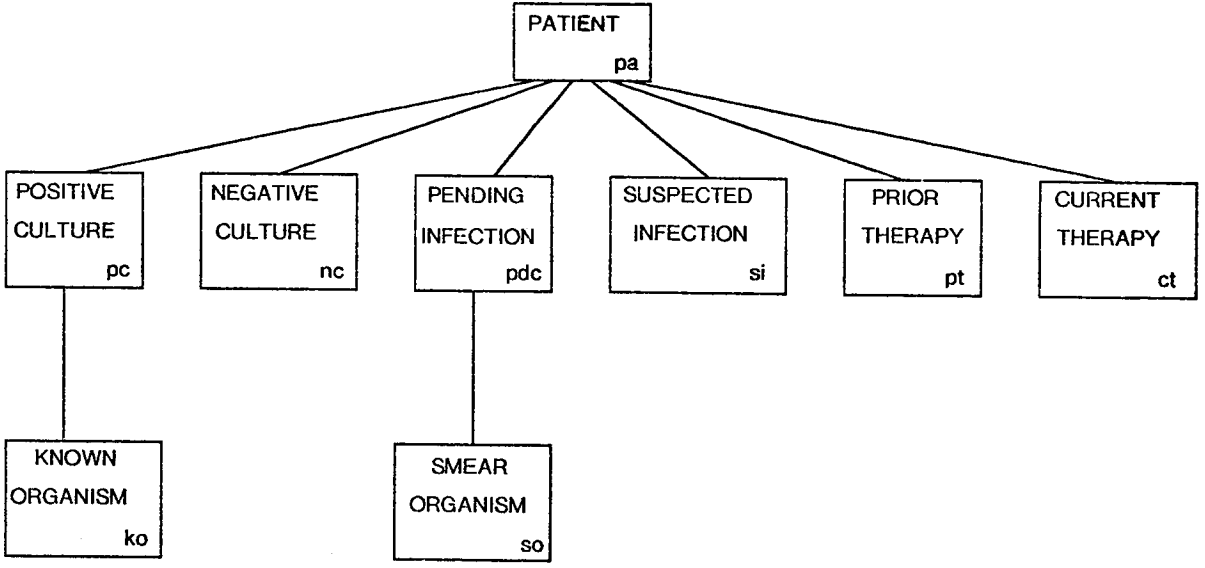

FIGURE 27-3 MYCIN's static tree.

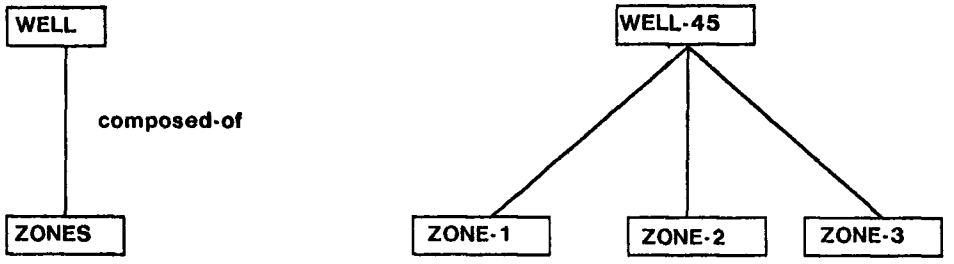

FIGURE 27-4 LITHO's static tree and an instance tree.

fixed by the tree of context-types. For this reason, physical objects, represented in this *part-whole fashion,* lend themselves more readily to the current context tree mechanism.

The last major use of the context tree, which is closely related to the part-whole use described above, has been to *represent important events or situations* that happen to an object. Thus, in the SACON system, a LOADING describes an anticipated scenario or maneuver (such as pounding or braking) to which the particular SUBSTRUCTURE is subjected. Each LOADING, in turn, is composed of a number of independent LOAD-COMPONENTS, distinguished by the direction and intensity of the applied force. Other uses of this organizational idea have been to represent individual past PREGNANCIES and current VISITS of a pregnant woman in the GRAVIDA system of Catanzarite (unpublished; see Figure 27-5) and the anticipated use of BLEEDING-EPISODES of a PATIENT in the CLOT system (Figure 27-6; see also Chapter 16).[4]

The primary reason for defining additional context-types in a consultant is to represent multiple instances of an entity during a case. Some users may like to define context-types that always have one instance and no more, primarily for purposes of organization, but this is often unnecessary (and even cumbersome).[5] For example, one might want to write rules that use various attributes of a patient's liver, but since there is always exactly one liver for a patient there is no need to have a liver context; any attribute of the liver can simply be viewed as an attribute of the patient.

Reference to parameters of contexts in *different* parts of an instance tree is currently very awkward. For example, in MYCIN, a particular drug may be associated somehow with a particular organism (Figure 27-7). However, this relationship between context-instances is *not* one that always holds

[4]It should be noted that use of the context mechanism to handle sequential visits in the GRAVIDA system is experimental and required the definition of numerous additional functions for this purpose. They are not currently in EMYCIN.

[5]Note, however, that separating unique concepts out into single contexts may provide more understandable rule translations due to the conventions of context-name substitutions in text generation. See Chapter 18 for further discussion of this point.

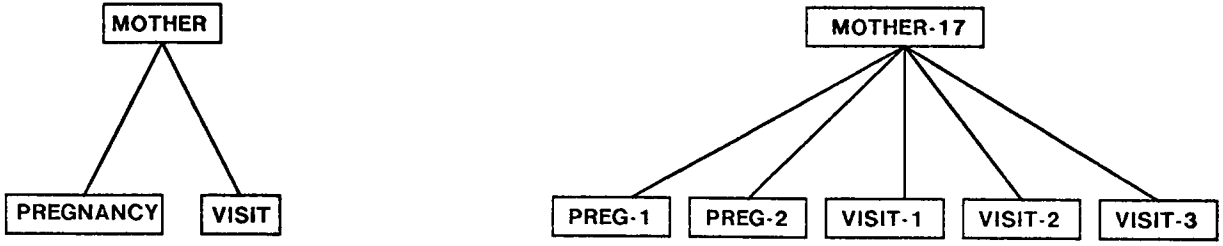

FIGURE 27-5 GRAVIDA's static tree and an instance tree.

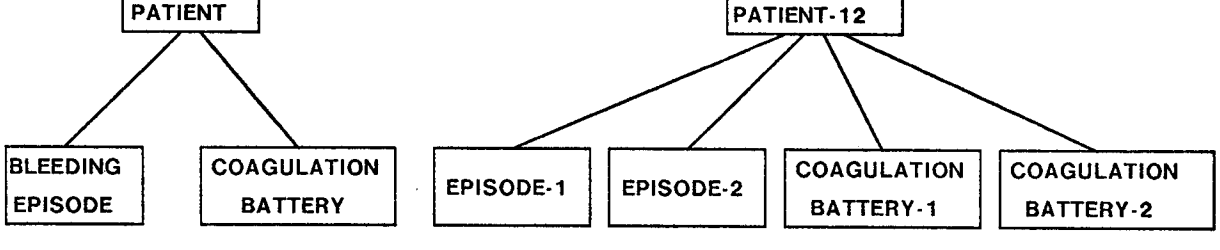

FIGURE 27-6 CLOT's static tree and an instance tree.

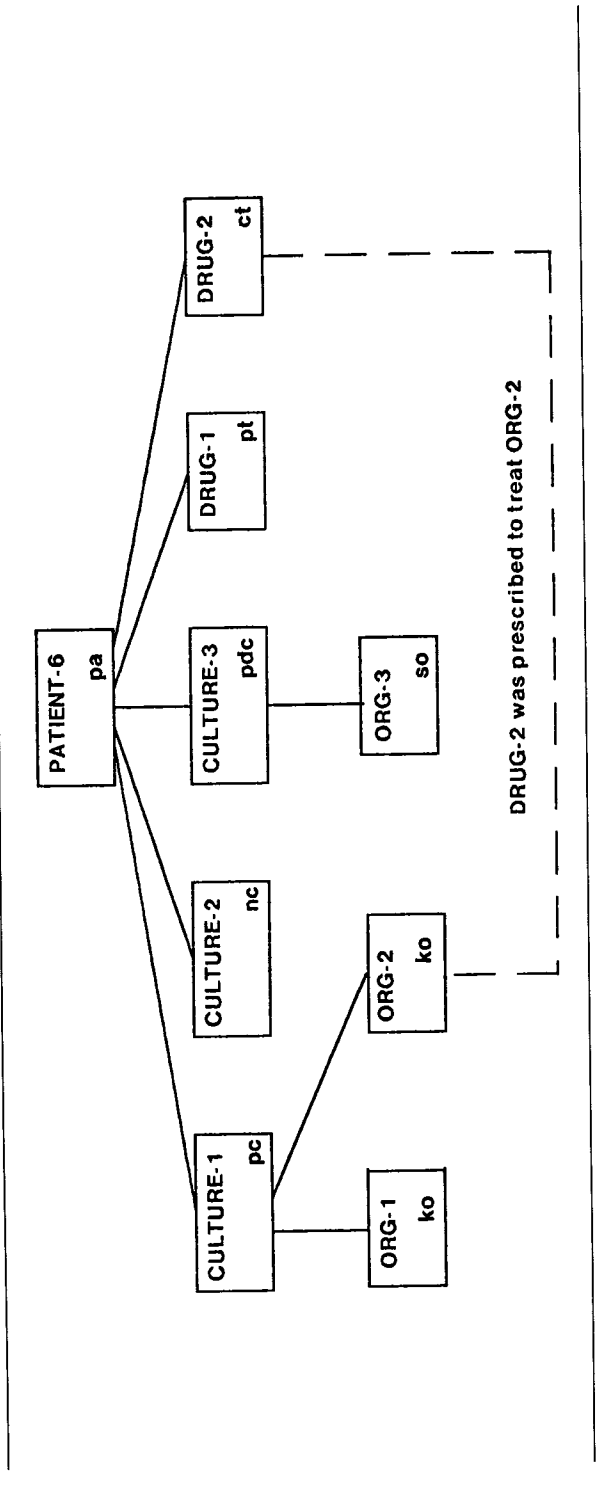

FIGURE 27-7 An example of a dynamic relationship between context-instances.

between all organisms and all drugs: not all drugs are prescribed to treat all identified organisms. This "prescribed for" relationship cannot be stated statically, independently of the case. Special predicate and action functions must be written to establish and manipulate these kinds of relationships between instances. It is best to avoid these interactions between disjoint parts of the tree during the initial design of the knowledge base.

Summing up our experience with this mechanism and considering its relative inflexibility, we offer this final caveat: for an initial system design, those using EMYCIN should start small and should use only one or two context-types. They should plan the structure of the consultant's context tree carefully before running the EMYCIN system, since restructuring a context tree is perhaps the most difficult and time-consuming knowledge-base construction task. Indeed, restructuring the context tree implies a complete restructuring of the rest of the knowledge base.

27.2 Grain Size of Rules

We had noticed that MYCIN's knowledge is "shallow" in the sense that its rules encode empirical associations but not theoretical laws. MYCIN lacks explicit representations of the "deep" understanding, such as an expert has, of causal mechanisms and reasoning strategies in medicine. MYCIN's rules do include some causal relations and definitions as well as structural relations, but all these are not cleanly separated from the heuristics and "compiled knowledge" that make up most of the rule set.

When we were building the initial system, we recognized that many rules were "broad-brush" treatments of complex processes, skipping from A to E in one leap and omitting any mention of B, C, and D in a chain such as A → B → C → D → E. We were focusing on rules whose "grain size" was of *clinical* significance. Even though finer-grained rules were often discussed, we consciously omitted them if the finer distinctions would not improve the program's ability to suggest appropriate treatments for infections or if they would not improve the understandability of the program for clinicians.[6] That is, the clinical significance of the conclusions determined the vocabulary of the rules. Thus, from the standpoint of performance, many causal mechanisms were not needed for reasoning from evidence to appropriate conclusions.

Examples of this collapsing of inference steps abound in all domains. For instance, physicians generally use a diuretic, such as furosemide, to treat edema or congestive heart failure without thinking twice about it. It is typically only when a patient fails to respond that the physician considers the mechanism of the drug's action in order to find, perhaps, another drug

[6]Note that physicians will be able to understand rules that medical students sometimes find confusing. See Chapter 20 for a further discussion of the grain size of rules.

to give with the first in order to produce the desired effect. Or, in a nonmedical domain, a mechanic often makes adjustments in response to manifestations of an automobile problem (e.g., adjusting the carburetor in response to stalling) and considers more detail only if the first few adjustments fail. An example from MYCIN is cited by Clancey in Chapter 29, in his discussion of the tetracycline rule: "If the patient is less than 8 years old, don't prescribe tetracycline." This rule lacks ties to the deeper understanding of drug action of which it is a consequence. Thus it is not only difficult for a student to remember, but also difficult for one to know how to modify or to know exactly how far the premise clause can be stretched safely.

We also recognized that many of the attributes mentioned in rules are not primitive observational terms in the same sense that values of laboratory tests are. For example, MYCIN asks whether a patient is getting better or worse in response to therapy, just as it asks for serum glucose levels. Obviously, there are a number of rules that could be written to infer whether the patient is better, mentioning such things as change in temperature, eating habits, and general coloring. That is, we chose a rule of the form A → B, with A as a primitive, rather than several rules in the following form:

$$
\begin{aligned}
&A_1 \rightarrow A \\
&A_2 \rightarrow A \\
&\vdots \\
&A_n \rightarrow A \\
&A \rightarrow B
\end{aligned}
$$

Neither of these shortcuts is a fatal flaw in the methodology of rule-based systems. Expanding the rule set to cover the richer knowledge physicians are known to hold would be possible, but time-consuming and unnecessary for improving MYCIN's advice in consultations. The consultation program, after all, was designed for use by physicians, and it seemed reasonable to leave some of the more basic observations up to them. However, as a result, there is considerable knowledge absent from MYCIN. As mentioned in Part Eight, successful tutoring depends on deep knowledge even more than successful consulting does.

27.3 Strategic, Structural, and Support Knowledge

The missing knowledge is of three classes: strategic, structural, and support. Strategic knowledge is an important part of expertise. MYCIN's built-in strategy is cautious: gather as much evidence as possible (without de-

manding new tests) for and against likely causes and then weigh the evidence. Operationally, this translates into exhaustive rule invocation whereby (a) all (relevant) rules are tried and (b) all rules whose left-hand sides match the case (and whose right-hand sides are relevant to problem-solving goals) have their right-hand sides acted upon. But under different circumstances, other strategies would be more appropriate. In emergencies, for example, physicians cannot take the time to gather much history data. Or, with recurring illness, physicians will order new tests and wait for the results. Deciding on the most appropriate strategy depends on medical knowledge about the context of the case. MYCIN's control structure is not concerned with resource allocation; it assumes that there is time to gather all available information that is relevant and time to process it. Thus MYCIN asks 20–70 questions and processes 1–25 rules between questions. We estimate that MYCIN executes about 50 rules per second (exclusive of I/O wait time). With larger amounts of data or larger numbers of rules, the control structure would need additional meta-rules that estimate the costs of gathering data and executing rules, in order to weigh costs against benefits. Also, in crisis situations or real-time data interpretation, the control structure would need to be concerned with the allocation of resources.[7]

One way to make strategic knowledge explicit is by putting it in *meta-rules*, as discussed in Chapter 28. They are rules of the same IF/THEN form as the medical rules, but they are "meta" in the sense that they talk *about* and reason *with* the medical rules. One of the interesting aspects of the meta-rule formalism, as Davis designed it, is that the same rule interpreter and explanation system work for meta-rules as for object-level rules. (Chapter 23 discussed the use of prototypes, or frames, for representing much of the same kind of knowledge about problem solving.) Making strategy knowledge explicit has come to be recognized as an important design consideration for expert systems (Barnett and Erman, 1982; de Kleer et al., 1977; Genesereth, 1981; Patil et al., 1981) because it can make a system's reasoning more efficient and more understandable.

Structural knowledge in medicine includes anatomical and physiological information about the structure and function of the body and its systems.[8] It is part of what we believe is needed for "deeper" reasoning about diagnosis. A structural model showing, *inter alia*, the normal connections of subparts can be used for reasoning about abnormalities. In contrast, representing this information in rules would force explicit mention of the

[7]In the AM and EURISKO programs (Lenat, 1976; 1983), Lenat has added information about maximum amounts of time to spend on various tasks, which keeps those programs from "overspending" computer time on difficult tasks of low importance. (EURISKO can also decide to change those time allocations.) In PROSPECTOR (Duda et al., 1978a), attention is focused on the rules that will add the most information, i.e., that will most increase or decrease the probability of the hypothesis being pushed. In Fox's system (Fox, 1981), the *estimated cost* of evaluating premises of rules helps determine which rules to invoke.

[8]More generally, we want to talk about the structure of any system or device we want an expert system to analyze, such as electronic circuits or automobiles.

abnormal situations and their manifestations. Thus there is a saving in the number of items represented explicitly in a rich structural model as opposed to an equally rich rule set. In medicine this point has been made by the Rutgers group (Kulikowski and Weiss, 1971) in the context of the CASNET program for diagnosing glaucomas. More recently, it is being advanced by Patil et al. (1981), Kunz (1983), Pople (1982), and others. In the domain of electronics almost everyone has noticed that a circuit diagram and causal knowledge are powerful pieces of knowledge to have [see, for example, Brown et al. (1974), Davis et al. (1982), Genesereth (1981), Grinberg, (1980)]. Structural knowledge also includes knowledge about the structure of the domain, e.g., the taxonomy of important concepts. This structure is an important reference point for guiding the problem solver in writing strategy rules.

Support knowledge includes items of information that are relevant for understanding a rule (or other knowledge structure). In early versions of MYCIN, we attached extra information to rules as justification for them or as historical traces of their evolution. For example, the *literature citations* provide credibility as well as pointers to more detailed information. The *names* of the persons who authored or edited a rule and the *dates* when it was created or edited are important pointers to persons responsible for the interpretation of the literature. The slot called "Justification" was created as a repository for the author's *comments* about why the rule was thought to be necessary in the first place. Additional support for a program's knowledge comes from deeper theoretical knowledge. Quantum chemistry, for example, could have been (but was not) referenced as support for DENDRAL's rules of mass spectrometry; pharmacology could have been (but was not) referenced to support MYCIN's rules of drug therapy. In general, support knowledge further explains the facts and relations of the domain knowledge. The contexts of tutoring and explanation demonstrate the need for support knowledge better than does the context of consultation because the additional support for rules is more relevant to *understanding* them than to *using* them (see Part Eight).

Recently, we have shifted our focus for this line of work from MYCIN to NEOMYCIN (Clancey and Letsinger, 1981), an updated version of the MYCIN knowledge base, representation, and control structure. In brief, it separates the diagnostic strategies clearly from the medical rules and facts used for diagnosing individual cases. By doing this, it can better serve as a basis for tutoring, as discussed in Chapter 26. NEOMYCIN was undertaken because of the issues noted in the following two chapters, but it is still too early to draw conclusions from the work.