15

EMYCIN: A Knowledge Engineer's Tool for Constructing Rule-Based Expert Systems

William van Melle, Edward H. Shortliffe, and Bruce G. Buchanan

Much current work in artificial intelligence focuses on computer programs that aid scientists with complex reasoning tasks. Recent work has indicated that one key to the creation of intelligent systems is the incorporation of large amounts of task-specific knowledge. Building knowledge-based, or expert, systems from scratch can be very time-consuming, however. This suggests the need for general tools to aid in the construction of knowledge-based systems.

This chapter describes an effective domain-independent framework for constructing one class of expert programs: rule-based consultants. The system, called EMYCIN, is based on the domain-independent core of the MYCIN program. We have reimplemented MYCIN as one of the consultation systems that run under EMYCIN.

15.1 The Task

EMYCIN is used to construct a *consultation program,* by which we mean a program that offers advice on problems within its domain of expertise. The consultation program elicits information relevant to the case by asking

This chapter is a shortened and edited version of a paper appearing in *Pergamon-Infotech state of the art report on machine intelligence,* pp. 249–263. Maidenhead, Berkshire, U.K.: Infotech Ltd., 1981.

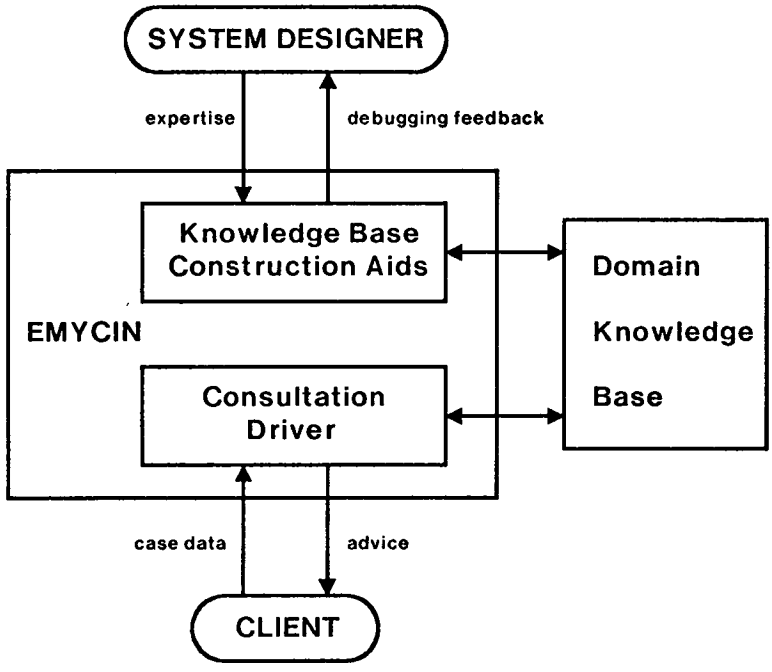

FIGURE 15-1 The major roles of EMYCIN: acquiring a knowledge base from the system designer, and interpreting that knowledge base to provide advice to a client.

questions. It then applies its knowledge to the specific facts of the case and informs the user of its conclusions. The user is free to ask the program questions about its reasoning in order to better understand or validate the advice given.

There are really two "users" of EMYCIN, as depicted in Figure 15-1. The *system designer,* or *expert,* interacts with EMYCIN to produce a *knowledge base* for the domain. EMYCIN then interprets this knowledge base to provide advice to the *client,* or *consultation user.* Thus the combination of EMYCIN and a specific knowledge base of domain expertise is a new *consultation program.* Some instances of such consultation programs are described below.

15.2 Background

Some of the earliest work in artificial intelligence attempted to create generalized problem solvers. Programs such as GPS (Newell and Simon, 1972)

and theorem provers (Nilsson, 1971), for instance, were inspired by the apparent generality of human intelligence and motivated by the desire to develop a single program applicable to many problems. While this early work demonstrated the utility of many general-purpose techniques (such as problem decomposition into subgoals and heuristic search in its many forms), these techniques alone did not offer sufficient power for high performance in complex domains.

Recent work has instead focused on the incorporation of large amounts of task-specific knowledge in what have been called *knowledge-based systems.* Such systems have emphasized high performance based on the accumulation of large amounts of knowledge about a single domain rather than on nonspecific problem-solving power. Some examples to date include efforts at symbolic manipulation of algebraic expressions (Moses, 1971), chemical inference (Lindsay et al., 1980), and medical consultations (Pople, 1977; Shortliffe, 1976). Although these systems display an expert level of performance, each is powerful in only a very narrow domain. In addition, assembling the knowledge base and constructing a working program for such domains is a difficult, continuous task that has often extended over several years. However, because MYCIN included in its design the goal of keeping the domain knowledge well separated from the program that manipulates the knowledge, the basic rule methodology provided a foundation for a more general rule-based system.

With the development of EMYCIN we have now come full circle to GPS's philosophy of separating the deductive mechanism from the problem-specific knowledge; however, EMYCIN's extensive user facilities make it a much more accessible environment for producing expert systems than were the earlier programs.[1] Like MYCIN's, EMYCIN's representation of facts is in attribute-object-value triples, with an associated certainty factor. Facts are associated in *production rules.* Rules of the same form are shown throughout this book. Figures 16-2 and 16-5 in the next chapter show rules from two different consultation systems constructed in EMYCIN.

15.2.1 Application of Rules—The Rule Interpreter

The control structure is primarily MYCIN's goal-directed backward chaining of rules. At any given time, EMYCIN is working toward the goal of establishing the value of some parameter of a context; this operation is termed *tracing* the parameter. To this end, the system retrieves the (precomputed) list of rules whose conclusions bear on the goal. SACON's Rule 50 (see Figures 15-2 and 16-2) would be one of several rules retrieved in an attempt to determine the stress of a substructure. Then for each rule

[1]Even so, it is still not an appropriate tool for building certain kinds of application systems because some of its power comes from the specificity of the rule-based representation and backward-chaining inference structure. See Section 15.5 for a discussion of these limitations.

in the list, EMYCIN evaluates the premise; if true, it makes the conclusion indicated in the action. The order of the rules in the list is assumed to be arbitrary, and all the rules are applied unless one of them succeeds and concludes the value of the parameter with certainty (in which case the remaining rules are superfluous).

This control structure was also designed to be able to deal gracefully with incomplete information. If the user is unable to supply some piece of data, the rules that need the data will fail and make no conclusions. The system will thus make conclusions, if possible, based on less information. Similarly, if the system has inadequate rules (or none at all) for concluding some parameter, it may ask the user for the value. When too many items of information are missing, of course, the system will be unable to offer sound advice.

15.2.2 More on the Rule Representation

There are many advantages to having rules as the primary representation of knowledge. Since each rule is intended to be a single "chunk" of information, the knowledge base is inherently modular, making it relatively easy to update. Individual rules can be added, deleted, or modified without drastically affecting the overall performance of the system. The rules are also a convenient unit for explanation purposes, since a single step in the reasoning process can be meaningfully explained by citing the English translation of the rule used.

While the syntax of rules permits the use of any LISP functions as matching predicates in the premises of rules, or as special action functions in the conclusions of rules, there is a small set of standard functions that are most frequently used. The system contains information about the use of these predicates and functions in the form of function *templates.* For example, the predicate SAME is described as follows:

(a) *function template:* (SAME CNTXT PARM VALUE)
(b) *sample function call:* (SAME CNTXT SITE BLOOD)

The system can use these templates to "read" its own rules. For example, the template shown here contains the standard symbols CNTXT, PARM, and VALUE, indicating the components of the associative triple that SAME tests. If clause (b) above appears in the premise of a given rule, the system can determine that the rule needs to know the site of the culture and, in particular, that it tests whether the culture site is (i.e., is the same as) blood. When asked to display rules that are relevant to blood cultures, the system will know that this rule should be selected. The most common matching predicates and conclusion functions are those used in MYCIN (see Chapter 5): SAME, NOTSAME, KNOWN, NOTKNOWN, DEFINITE, NOTDEFINITE, etc.

15.2.3 Explanation Capability

As will be described in Part Six, EMYCIN's *explanation program* allows the user of a consultation program to interrogate the system's knowledge, either to find out about inferences made (or not made) during a particular consultation or to examine the static knowledge base in general, independently of any specific consultation.

During the consultation, EMYCIN can offer explanations of the current, past, and likely future lines of reasoning. If the motivation for any question that the program asks is unclear, the client may temporarily put off answering and instead inquire why the information is needed. Since each question is asked in an attempt to evaluate some rule, a first approximation to an explanation is simply to display the rule currently under consideration. The program can also explain what reasoning led to the current point and what use might later be made of the information being requested. This is made possible by examining records left by the rule interpreter and by reading the rules in the knowledge base to determine which are relevant. This form of explanation requires no language understanding by the program; it is invoked by simple commands from the client (WHY and HOW).

Another form of explanation is available via the *Question-Answering (QA) Module,* which is automatically invoked after the consultation has ended, and which can also be entered during the consultation to answer questions other than those handled by the specialized WHY and HOW commands mentioned above. The QA Module accepts simple English-language questions (a) dealing with any conclusion drawn during the consultation, or (b) about the domain in general. Explanations are again based on the rules; they should be comprehensible to anyone familiar with the domain, even if that person is not familiar with the intricacies of the EMYCIN system. The questions are parsed by pattern matching and keyword look-up, using a dictionary that defines the vocabulary of the domain. EMYCIN automatically constructs the dictionary from the English phrases used in defining the contexts and parameters of the domain; the system designer may refine this preliminary dictionary to add synonyms or to fine-tune QA's parsing.

15.3 The System-Building Environment

The system designer's principal task is entering and debugging a knowledge base, viz., the rules and the object-attribute structures on which they operate. The level at which the dialogue between system and expert takes place is an important consideration for speed and efficiency of acquisition.

```
IF:  Composition = (LISTOF METALS) and
     Error < 5 and
     Nd-stress > .5 and
     Cycles > 10000
THEN: Ss-stress = fatigue
```

FIGURE 15-2 Example of ARL format for SACON's Rule 50.

The knowledge base must eventually reside in the internal LISP format that the system manipulates to run the consultation and to answer questions. At the very basic level, one could imagine a programmer using the LISP editor to create the necessary data structures totally by hand;[2] here the entire translation from the expert's conceptual rule to LISP data structures is performed by the programmer. At the other extreme, the expert would enter rules in English, with the entire burden of understanding placed on the program.

The actual means used in EMYCIN is at a point between these extremes. Entering rules at the base LISP level is too error-prone, and requires greater facility with LISP on the part of the system designer than is desirable. On the other hand, understanding English rules is far too difficult for a program, especially in a new domain where the vocabulary has not even been identified and organized for the program's use. (Just recognizing new parameters in free English text is a major obstacle.[3]) EMYCIN instead provides a terse, stylized, but easily understood, language for writing rules and a high-level knowledge base editor for the knowledge structures in the system. The knowledge base editor performs extensive checks to catch common input errors, such as misspellings, and handles all necessary bookkeeping chores. This allows the system builder to try out new ideas quickly and thereby to get some idea of the feasibility of any particular formulation of the domain knowledge into rules.

15.3.1 Entering Rules

The Abbreviated Rule Language (ARL) constitutes an intermediate form between English and pure LISP. ARL is a simplified ALGOL-like language that uses the names of the parameters and their values as operands; the operators correspond to EMYCIN predicates. For example, SACON's Rule 50 could have been entered or printed as shown in Figure 15-2.

ARL resembles a shorthand form derived from an *ad hoc* notation that we have seen several of our domain experts use to sketch out sets of rules.

[2]This is the way the extensive knowledge base for the initial MYCIN system was originally created.

[3]The task of building an assistant for designers of new EMYCIN systems is the subject of current research by James Bennett (Bennett, 1983). The name of the program is ROGET.

The parameter names are simply the labels that the expert uses in defining the parameters of the domain. Thus they are familiar to the expert. The conciseness of ARL makes it much easier to enter than English or LISP, which is an important consideration when entering a large body of rules.

Rule Checking

As each rule is entered or edited, it is checked for syntactic validity to catch common input errors. By syntactic, we mean issues of rule form—whether terms are spelled correctly, values are legal for the parameters with which they are associated, etc.—rather than the actual information content (i.e., semantic considerations as to whether the rule "makes sense"). Performing the syntactic check at acquisition time reduces the likelihood that the consultation program will fail due to "obvious" errors, thus freeing the expert to concentrate on debugging logical errors and omissions. These issues are also discussed in Chapter 8.

EMYCIN's purely syntactic check is made by comparing each clause with the corresponding function template and seeing that, for example, each PARM slot is filled by a valid parameter and that its VALUE slot holds a legal value for the parameter. If an unknown parameter is found, the checker tries to correct it with the Interlisp spelling corrector, using a spelling list of all parameters in the system. If that fails, it asks if this is a new (previously unmentioned) parameter. If so, it defines the new parameter and, in a brief diversion, prompts the system builder to describe it. Similar action is also taken if an illegal value for a parameter is found.

A limited semantic check is also performed: each new or changed rule is compared with any existing rules that conclude about the same parameter to make sure it does not directly contradict or subsume any of them. A contradiction occurs when two rules with the same set of premise clauses make conflicting conclusions (contradictory values or CF's for the same parameter); subsumption occurs when one rule's premise is a subset of another's, so that the first rule succeeds whenever the second one does (i.e., the second rule is more specific), and both conclude about the same values. In either case, the interaction is reported to the expert, who may then examine or edit any of the offending rules.

15.3.2 Describing Parameters

Information characterizing the parameters and contexts of the domain is stored as *properties* of each context or parameter being described. When a new entity is defined, the acquisition routines automatically prompt for the properties that are always needed (e.g., EXPECT, the list of values expected for this parameter); the designer may also enter optional properties (those

needed to support special EMYCIN features). The properties are all checked for validity, in a fashion similar to that employed by the rule checker.

15.3.3 System Maintenance

While the system designer builds up the domain knowledge base as described above, EMYCIN automatically keeps track of the changes that have been made (new or changed rules, parameters, etc.). The accumulated changes can be saved on a file by the system builder either explicitly with a simple command or automatically by the system every n changes (the frequency of automatic saving can be set by the system builder). When EMYCIN is started in a subsequent session, the system looks for this file of changes and loads it in to restore the knowledge base to its previous state.

15.3.4 Human Engineering

Although the discussion so far has concentrated on the acquisition of the knowledge base, it is also important that the resulting consultation program be pleasing in appearance to the user. EMYCIN's existing human-engineering features relieve the system builder of many of the tedious cosmetic concerns of producing a usable program. Since the main mode of interaction between the consultation program and the client is in the program's questions and explanations, most of the features concentrate on making that interface as comfortable as possible. A main feature in this category that has already been described is the explanation program—the client can readily find out why a question is being asked, or how the program arrived at its conclusions. The designer can also control, by optionally specifying the PROMPT property for each parameter that is asked for, the manner in which questions are phrased. More detail can be specified, for example, than would appear in a simple prompt generated by the system from the parameter's translation.

EMYCIN supplies a uniform input facility that allows the normal input-editing functions—character, word, and line deletions—and on display terminals allows more elegant editing capabilities (insertion or deletion in the middle of the line, for example) in the style of screen-oriented text editors. It performs spelling correction and TENEX-style completion[4] from a list of possible answers; most commonly this list is the list of legal

[4]After the user types ESCAPE or ALTMODE, EMYCIN fills out the rest of the phrase if the part the user has typed is unambiguous. For example, when EMYCIN expects the name of an organism, PSEU is unambiguous for PSEUDOMONAS-AERUGINOSA. Thus the automatic completion of input can save considerable effort and frustration.

values for the parameter being asked about, as supplied by the system designer.

In most places where EMYCIN prompts for input, the client may type a question mark to obtain help concerning the options available. When the program asks for the value of a parameter, EMYCIN can provide simple help by listing the legal answers to the question. The system designer can also include more substantial help by giving rephrasings of or elaborations on the original question; these are simply entered via the data base editor as an additional property of the parameter in question. This capability provides for both streamlined questions for experienced clients and more detailed explanations of what is being requested for those who are new to the consultation program.

15.3.5 Debugging the Knowledge Base

There is more to building a knowledge base than just entering rules and associated data structures. Any errors or omissions in the initial knowledge base must be corrected in the debugging process. In EMYCIN the principal method of debugging is to run sample consultations; i.e., the expert plays the role of a client seeking advice from the system and checks that the correct conclusions are made. As the expert discovers errors, he or she uses the knowledge acquisition facilities described above to modify existing rules or add new ones.

Although the explanation program was designed to allow the consultation user to view the program's reasoning, it is also a helpful high-level debugging aid for the system designer. Without having to resort to LISP-level manipulations, it is possible to examine any inferences that were made, find out why others failed, and thereby locate errors or omissions in the knowledge base. The TEIRESIAS program developed the WHY/HOW capability used in EMYCIN for this very task (see Chapter 9).

EMYCIN provides a debugger based on a portion of the TEIRESIAS program. The debugger actively guides the expert through the program's reasoning chain and locates faulty (or missing) rules. It starts with a conclusion that the expert has indicated is incorrect and follows the inference chain back to locate the error.

The rule interpreter also has a debugging mode, in which it prints out assorted information about what it is doing: which rules it tries, which ones succeed (and what conclusions they make), which ones fail (and for what reason), etc. If the printout indicates that a rule succeeded that should have failed, or vice versa, the expert can interrupt immediately, rather than waiting for the end of the consultation to do the more formal TEIRESIAS-style review.

In either case, once the problem is corrected, the expert can then restart and try again, with the consultation automatically replayed using the new or modified rules.

Case Library

EMYCIN has facilities for maintaining a library of sample cases. These can be used for testing a complete system, or for debugging a growing one. The answers given by the consultation user to all the questions asked during the consultation are simply stored away, indexed by their context and parameter. When a library case is rerun, answers to questions that were previously asked are looked up and automatically supplied; any new questions resulting from changes in the rule base are asked in the normal fashion. This makes it easy to check the performance of a new set of rules on a "standard" case. It is especially useful during an intensive debugging session, since the expert can make changes to the knowledge base and, with a minimum of extra typing, test those changes—effectively reducing the "turnaround time" between modifying a rule and receiving consultation feedback.

The BATCH Program

A problem common to most large systems is that new knowledge entered to fix one set of problems often introduces new bugs, affecting cases that once ran successfully. To simplify the task of keeping the knowledge base consistent with cases that are known to be correctly solved, EMYCIN's BATCH program permits the system designer to run any or all cases in the library in background mode. BATCH reports the occurrence of any changes in the results of the consultation and invokes the QA Module to explain why the changes occurred. Of course, the system builder must first indicate to the system which parameters represent the results or the most important intermediate steps by which the correctness of the consultation is to be judged. The use of the BATCH program could be viewed as a form of additional semantic checking to supplement the checking routinely performed at the time of rule acquisition.

15.3.6 The Rule Compiler

To improve efficiency in a running consultation program, EMYCIN provides a *rule compiler* that transforms the system's production rules into a decision tree, eliminating the redundant computation inherent in a rule interpreter. The rule compiler then compiles the resulting tree into machine code. The consultation program can thereby use an efficient deductive mechanism for running the actual consultation, while the flexible rule format remains available for acquisition, explanation, and debugging. For details about the rule compiler see van Melle (1980).

15.4 Applications

Several consultation systems have been written using EMYCIN. The original MYCIN program provides advice on diagnosis and therapy for infectious diseases. MYCIN is now implemented in EMYCIN, but its knowledge base was largely constructed before EMYCIN was developed as a separate system. SACON and CLOT (described in Chapter 16), PUFF (Aikins et al., 1983), HEADMED (Heiser et al., 1978), LITHO (Bonnet, 1981), DART (Bennett and Hollander, 1981), BLUEBOX (Mulsant and Servan-Schreiber, 1983), and several other demonstration systems have been successfully built in EMYCIN. All have clearly shown the power of starting with a well-developed framework and concentrating on the knowledge base. For example, to bring the SACON program to its present level of performance, about two person-months of the experts' time were required to explicate their task as consultants and to formulate the knowledge base, and about the same amount of time was required to implement and test the rules in a preliminary version of EMYCIN. CLOT was constructed as a joint effort by an experienced EMYCIN programmer and a collaborating medical student. Following approximately ten hours of discussion about the contents of the knowledge base, they entered and debugged in another ten hours a preliminary knowledge base of some 60 rules using EMYCIN. Both knowledge bases would need considerable refinement before the programs would be ready for general use. The important point, however, is that starting with a framework like EMYCIN allows system builders to focus quickly on the expertise necessary for high performance because the underlying framework is ready to accept it.

15.5 Range of Applicability

EMYCIN is designed to help build and run programs that provide consultative advice. The resulting consultation system takes as input a body of measurements or other information pertinent to a case and produces as output some form of recommendation or analysis of the case. The framework seems well suited for many diagnostic or analytic problems, notably some classes of fault diagnosis, where several input measurements (symptoms, laboratory tests) are available and the solution space of possible diagnoses can be enumerated. It is less well suited for "formation" problems, where the task is to piece together existing structures according to specified constraints to generate a solution.

EMYCIN was not designed to be a general-purpose representation language. It is thus wholly unsuited for some problems. The limitations

derive largely from the fact that EMYCIN has chosen one basic, readily understood representation for the knowledge in a domain: production rules that are applied by a backward-chaining control structure and that operate on data in the form of associative triples. The representation, at least as implemented in EMYCIN, is unsuitable for problems of constraint satisfaction, or those requiring iterative techniques.[5] Among other classes of problems that EMYCIN does not attempt to handle are simulation tasks and tasks involving planning with stepwise refinement. One useful heuristic in thinking about the suitability of EMYCIN for a problem is that the consultation system should work with a "snapshot" of information about a case. Good advice should not depend on analyzing a continued stream of data over a time interval.

Even those domains that have been successfully implemented have demonstrated some of the inadequacies of EMYCIN. In addition to representational difficulties, other problems noted have been the lack of user control over the consultation dialogue (e.g., the order of questions) and the amount of time a user must spend supplying information. These limitations are discussed further in subsequent chapters.

[5]The VM program (Chapter 22), however, has shown that production rules can be used to provide advice in a dynamic setting where iterative monitoring is required. Greatly influenced by EMYCIN design issues, VM deals with the management of patients receiving assisted ventilation after cardiac surgery.