PART SEVEN

Using Other Representations

21

Other Representation Frameworks

Representing knowledge in an AI program means choosing a set of conventions for describing objects, relations, and processes in the world. One first chooses a conceptual framework for thinking about the world—symbolically or numerically, statically or dynamically, centered around objects or around processes, and so forth. Then one needs to choose conventions within a given computer language for implementing the concepts. The former is difficult and important; the latter is both less difficult and less important because good programmers can find ways of working with almost any concept within almost any programming language.

In one respect finding a representation for knowledge is like choosing a set of data structures for a program to work with. Tables of data, for example, are often conveniently represented as arrays. But manipulating knowledge structures imposes additional requirements. Because some of an expert's knowledge is inferential, conventions are needed for a program to interpret the structures. And, as we have emphasized, an expert (or knowledge engineer) needs to be able to edit knowledge structures quickly and easily in order to refine the program's knowledge base iteratively. Some programming conventions facilitate editing and interpreting knowledge; others throw up road blocks.

The question of how to represent knowledge for intelligent use by programs is one of two major questions motivating research in AI. (The other major theme over the last 25 years is how to *use* the knowledge for intelligent problem solving.) Although we were not developing new representations in MYCIN, we were experimenting with the power of one representation, modified production rules, for reasoning in a detailed and ill-structured domain, medicine. Chapters 1 and 3 have described much of the historical context of our work with rules. As should be obvious from Chapters 3 through 6, we added many embellishments to the basic production rule representation in order to cope with the demands of the problem and of physicians. We stumbled over many items of medical knowledge that were difficult to encode or use in the simple formalism

with which we started. Our choice of rules and fact triples, with CF's, has been explained in Part Two. As summarized at the end of Chapter 3, we were under no illusion that we were creating a "pure" production system. We had taken many liberties with the formalism in order to make it more flexible and understandable. However, we still felt that the stylized condition-action form of knowledge brought many advantages because of its simplicity. For example, creating English translations from the LISP rules and translating stylized English rules into LISP were both somewhat simplified because of the restricted syntax. Similarly, creating explanations of a line of reasoning was simplified as well, because of the simple backward-chaining control structure that links rules together dynamically.

Representing knowledge in procedures was one alternative we were trying hard to avoid. Our experience with DENDRAL and with the therapy algorithm in MYCIN (Chapter 6) showed how inflexible and opaque a set of procedures could be for an expert maintaining a knowledge base. And, as mentioned in previous chapters, we saw that production rules offered some opportunity for making a knowledge base easier to understand and modify.

We were aware of predicate calculus as a possibility for representing MYCIN's knowledge. We were working in a period in AI research when logic and resolution-based theorem provers were being recommended for many problems. We did not seriously entertain the idea of using logic, however, largely because we felt that inexact reasoning was undeveloped in theorem-proving systems.

We had initially experimented with a semantic network representation, as mentioned in Chapter 3. Although we felt we could store medical knowledge in that form, we felt it was difficult to focus a dialogue in which gaps in the knowledge were filled both by inference and by the user's answers to questions. Minsky's paper on frames (Minsky, 1975) did not appear until after this work was well underway. Even so, we were looking for a more structured representation, specifically rules, to build editors and parsers for, to modify and explain, and to reason with in an understandable line of reasoning.

In this part we describe three experiments with alternative representations and control structures in programs called VM, CENTAUR, and WHEEZE. The first two programs were written for Ph.D. requirements, the last as a class project. All are programs that work on medical problems, although in areas outside of infectious diseases. Another experiment with representations is described in Chapter 20 in the context of explanation. There MYCIN's rules are rewritten in an inference net (cf. Duda et al., 1978b) in order to facilitate explaining the inferences at different levels of detail.

The VM program discussed in Chapter 22 was selected by Professor E. Feigenbaum, H. Penny Nii, and Dr. John Osborn and worked on primarily by Larry Fagan for his Ph.D. dissertation. Feigenbaum and Nii had

been developing the SU/X program[1] (Nii and Feigenbaum, 1978) for interpretation of multisensor data. Feigenbaum was a friend of Osborn's, knew of Osborn's pioneering work on computer monitoring in intensive care, and saw this as a possible domain in which to explore further the problems in multisensor signal understanding involving signals for which the time course is important to the interpretation. Osborn agreed to be the expert collaborator. Fagan had been working on MYCIN and had contributed to the code as well as to the knowledge base of meningitis rules. (In Feigenbaum's words, Fagan had become "MYCINized.") So it was natural that his initial thinking about the ICU data interpretation problem was in MYCIN's terms. Fagan quickly found, however, that the MYCIN model was not appropriate for a problem of monitoring data continuously over time. MYCIN was much too oriented toward a "snapshot" of data about a patient at a fixed time (although some elements of data in the "snapshot" name historical parameters, such as dates of prior infections). The only obvious mechanism for making MYCIN work with a stream of data in the ICU was to restart the program at frequent time intervals to reason about each new "snapshot" of data gathered during each 2–5 minute time period. This is inelegant and completely misses any sense of continuity or the changing context in which data are being gathered. Thus VM was designed to remedy this deficiency.

The other two programs in Part Seven were designed as alternatives to a rule-based representation, varying the representation of one program, called PUFF. Although desirable, it is difficult in AI to experiment with programs by varying one parameter at a time while holding everything else fixed. Of course, not everything else could remain fixed for such a gross experiment. Both CENTAUR and WHEEZE, discussed in Chapters 23 and 24, were deliberate attempts to alter the representation and control of the PUFF program (while leaving the knowledge base unchanged) in order to examine advantages and disadvantages of alternatives.

PUFF is a program that diagnoses pulmonary (lung) diseases. The problem was suggested to Feigenbaum and Nii by Osborn at the time VM was being formulated, and appeared to be appropriate for a MYCIN-like approach. It was initially programmed using EMYCIN (see Part Five), in collaboration with Drs. R. Fallat and J. Osborn at Pacific Medical Center in San Francisco (Aikins et al., 1983). About 50–60 rules were added to EMYCIN [in a much shorter time than expected (Feigenbaum, 1978)] to interpret the type and severity of pulmonary disorders.[2] The primary data are mostly from an instrument known as a spirometer that measures flows and volumes of patients' inhalation and exhalation. The conlusions are diagnoses that account for the spirometer data, the patient history data, and the physician's observations.

---

[1]*Later known as HASP* (Nii et al., 1982).

[2]These handled obstructive airways disease. Many other rules were later added to handle other classes of pulmonary disease. The system now contains about 250 rules.

EMYCIN-PUFF	(Aikins and Nii—see Chapter 14)
CENTAUR	(Aikins—see Chapter 23)
WHEEZE	(Smith and Clayton—see Chapter 24)
BASIC-PUFF	(Pacific Medical Center—see Aikins et al., 1983)
AGE-PUFF	(Nii and Aiello—see Aiello and Nii, 1981)

FIGURE 21-1 Five implementations of PUFF.

PUFF has been a convenient vehicle for experimentation because it is a small system. Figure 21-1 lists five different implementations of essentially the same knowledge base.

In developing CENTAUR, Aikins focused on the problem of making control knowledge explicit and understandable. She recognized the awkwardness of explanations of rules or rule clauses that were primarily *controlling* MYCIN's inferences as opposed to *making* substantive inferences. For example, many of the so-called self-referencing rules are awkward to explain:

If A & B & C,
then A

In these rules, one intent of mentioning parameter A in both conclusion and premise is to *screen* the rule and keep it from forcing questions about parameters B and C if there is not already evidence for A. This is largely an issue of control, and the kind of problem that CENTAUR is meant to remedy. The solution is to use frames to represent the context and control information and MYCIN-like rules to represent the substantive medical relations. Thus there is a frame for A to represent the context in which a set of rules should be invoked, one of which would be:

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

This is much more natural to explain than trying to say why, or in what sense, A can be evidence for itself. CENTAUR was demonstrated using the same knowledge as in the EMYCIN version of PUFF (Aikins, 1983).

David Smith and Jan Clayton developed WHEEZE as a further experiment with frames. They asked, in effect, if *all* the knowledge in PUFF could be represented in frames and what benefits would follow from doing so. In a short time (as a one-term class project) they reimplemented PUFF with a frame-based representation. Chapter 24 is a summary of their results.

The version of PUFF written in BASIC (BASIC-PUFF) is a simplified version of the EMYCIN rule interpreter with the medical knowledge built into the code (Aikins et al., 1983). It was redesigned to run efficiently on

a PDP-11 in the pulmonary laboratory at Pacific Medical Center. Its knowledge has been more finely tuned than it was in the original version, but is largely the same. BASIC-PUFF is directly coupled to the spirometer in the pulmonary function lab and automatically provides interpretations of the test results. Thus it turns the spirometer into a "smart instrument" instead of simply a data-collecting and recording device. Its interpretations are printed immediately, reviewed by a physician, and inserted into the permanent record with the physician's signature. In the majority of cases, the physician makes no additions or corrections to the conclusions; in some, however, additional notes are made to clarify the program's suggestions. BASIC-PUFF provides one model of technology transfer for expert systems: first implement a prototype with "off-the-shelf" tools such as EMYCIN, then rewrite the system to run efficiently on a small computer.

Another experiment in which the PUFF knowledge base was recast into a different formalism is the AGE-PUFF version (Aiello and Nii, 1981). The intent was to use this small, easily managed knowledge base to experiment with control issues, more specifically to explore the adequacy of the BLACKBOARD model, with event-driven control (Erman et al., 1980). Further experiments with AGE-PUFF are reported by Aiello (1983).

One of the difficulties with a production rule formalism is in representing control information. For example, if we want rules R3, R5, and R7 to be executed in that order, then we have to arrange for the LHS of R7 not to match any current data base until after R3 and R5 have fired. Often this is accomplished by defining a flag that is set when and only when R3 fires and that is checked by R5, and another that is set by R5 and checked by R7, as described in Chapter 2. The authors of MYCIN's rules have only a few means available to influence the system's backward chaining, one of which is to define "dummy" parameters that act as flags. To the best of our knowledge, this was not done in MYCIN (in fact, it was explicitly avoided), but it has been done by others using EMYCIN.

Another means of influencing the control is to order the clauses in premises of rules. This was done much of the time as a way of keeping MYCIN from pursuing minutiae before the more general context that motivates asking about minute details was established. Since MYCIN evaluates the premise clauses from first to last, in order,[3] putting more general, context-setting clauses at the beginning of the premise assures that the more specific clauses will not be asked about, or even considered, unless the context is appropriate. Using the order of premise clauses for this kind of screening permits the system builder to use early clauses to ensure that some parameters are traced first. For example, the predicate KNOWN is often used to cause a parameter to be traced.

Still another means of representing controlling information in the rule-based formalism is via meta-rules, described in Chapter 28. Another

[3]An exception is the preview mechanism described earlier.

similar approach is via strategy rules, as described in Chapter 29. The unity path mechanism (Chapter 3) also affects the order of rule invocation.

ONCOCIN (discussed in Chapters 32 and 35) incorporates many of the ideas from these experiments, most notably the framelike representation of control knowledge and the description of changing contexts over time. It builds on other results presented in this book as well, so its design is described later. ONCOCIN clearly shows the influence of the evolution of our thinking presented in this section.

One piece of recent research not included in this volume is the rerepresentation of MYCIN's knowledge along the lines described in Chapter 29. The new program, called NEOMYCIN (Clancey and Letsinger, 1981), carries much of its medical knowledge in rules. But it also represents (a) the taxonomy of diseases as a separate hierarchy, (b) strategy knowledge as meta-rules, (c) causal knowledge as links in a network, and (d) knowledge about disease processes in the form of frames characterizing location and temporal properties. One main motivation for the reconceptualization was to provide improved underpinnings for the tutorial program described in Chapter 26. Because of the richer knowledge structures in NEOMYCIN, informative explanations can be given regarding the program's diagnostic strategies, as well as the medical rules.

NEOMYCIN, along with other recent work, emphasizes the desirability of augmenting MYCIN's homogeneous set of rules with a classification of types of knowledge and additional knowledge of each type. In MYCIN's rule set, the causal mechanisms, the taxonomic structure of the domain, and the problem-solving strategies are all lumped together. An augmented knowledge base should separate these different types of knowledge to facilitate explanation and maintenance of the knowledge base, and perhaps to enhance performance as well. Causal mechanisms have been represented and used in several domains, including medicine (Patil et al., 1981) and electronics debugging (Davis, 1983). Mathematical models have been merged with symbolic causal models in AI/MM (Kunz, 1983). As a result of this recent work, considerably richer alternatives than MYCIN's homogeneous rule set can be found.

Finally, it should be noted that the chapters in this part describe rather fundamental viewpoints on representation. Within a rule-based or frame-based (or mixed) framework there are still numerous details of representing uncertainty, quantified variables, strategies, temporal sequences, bookkeeping information, and other concepts mentioned throughout the book.