Chapter 3

Consultation System

3.1 Introduction

In this and the succeeding two chapters MYCIN's implementation is presented in considerable detail. My goal is both to explain the data and control structures used by the program and to describe some of the complex and often unexpected problems that arose during system implementation. Less detailed discussions, which provide a general overview of the material in Chapters 3 and 5, may be found elsewhere [Shortliffe, 1973, 1975b]. In Chapter 2 the motivation behind many of MYCIN's capabilities was explained. If you bear those design criteria in mind throughout the remainder of this text, you will see the important role they have played.

This chapter specifically describes the Consultation System (Subprogram 1). As indicated in Figure 1-1, this subprogram uses both system knowledge from the corpus of rules, plus patient data entered by the physician, in order to generate advice for the user. Furthermore, the program maintains a dynamic data base that provides an ongoing record of the current consultation. As a result, this chapter must discuss both the nature of the various data structures and how they are used or maintained by the Consultation System.

Section 3.2 describes the corpus of rules and the associated data structures. It begins by looking at other rule-based systems and proceeds to a formal description of the rules used by MYCIN. Our quantitative truth model is briefly introduced and the mechanism for rule evaluation is explained. This section also describes the clinical parameters with which MYCIN is familiar and which form the basis for the conditional expressions in the PREMISE of a rule.

In § 3.3 MYCIN's goal-oriented control structure is described; mechanisms for rule invocation and question selection are explained at that time. The section also discusses the creation of the dynamic data base that is the foundation for both the system's advice and its explanation capabilities as described in Chapter 5.

Section 3.4 is devoted to an explanation of the program's context tree, i.e., the network of interrelated organisms, drugs, and cultures that characterize the patient and his current clinical condition. The need for such a data structure is clarified and the method for propagation (growth) of the tree is described.

As discussed in § 1.5.1, the final tasks in MYCIN's clinical problem area are the identification of potentially useful drugs and the selection of the best drug or drugs from that list. MYCIN's mechanism for making these decisions is discussed in § 3.5.

Section 3.6 discusses MYCIN's mechanisms for storing patient data and for permitting a user to changes the answer to a question. As will be described, these two capabilities are closely interrelated.

In § 3.7, I briefly mention some contemplated future extensions to the system. The concluding section then summarizes the advantages of the MYCIN approach, making comparisons with previous work in both AI and medical decision making.

3.2 System Knowledge

3.2.1 DECISION RULES

Automated problem-solving systems use criteria for drawing conclusions that often support a direct analogy to the rule-based knowledge representation used by MYCIN. Consider, for example, the conditional probabilities that underlie the Bayesian diagnosis programs discussed in § 1.3.4. Each probability statement provides information that may be expressed in an explicit rule format:

$$P(h|e) = X$$

means:

IF: E IS KNOWN TO BE TRUE
THEN: CONCLUDE THAT H IS TRUE WITH PROBABILITY X

The advantages of an explicit rule format are discussed in Section 4.3.

It is important to note, however, that the concept of rule-based knowledge is not unique, even for medical decision making programs. As will be explained, MYCIN's innovation rests with its novel application of representation techniques and goal-oriented control structures that have been developed by AI researchers. The contributions of the program to AI and medical decision making are summarized in Chapter 7.

3.2.1-1 Previous Rule-Based Systems

The need for representation of knowledge in IF-THEN format so pervades problem-solving in AI that many AI programs can be interpreted as rule-based systems once we recognize that all deductive or inferential statements are, in effect, decision rules. In fact, several of the new AI languages have provided data structures and control structures based on rules (theorems) [Bobrow, 1973]. For example, PLANNER [Hewitt, 1969, 1971, 1972] provides a formalism for the statement of theorems such as:

```
(CONSEQUENT
     (PART $?X $?Z)
     (GOAL (PART $?X $?Y))
     (GOAL (PART $?Y $?Z)))
```

This theorem simply states, in rule form, that:

```
IF:    YOU CAN FIND AN X THAT IS PART OF A Y, AND
       YOU CAN FIND A Z SUCH THAT THE Y IS PART OF THE Z
THEN:  YOU CAN CONCLUDE THAT THE X IS PART OF THE Z
```

Although there are several examples of AI programs that use some variety of rule-based knowledge, only four representative cases will be introduced here. The control structures used for processing the "rules" in these systems are not discussed until § 3.3.1.

The first example is the theorem-proving question–answering program named QA3 [Green, 1969]. As was pointed out in the example from PLANNER above, a theorem may be considered a rule. Green

states his rules in the predicate calculus. For example:

[1] (FA (X) (IF (IN X FIREMEN) (OWNS X RED-SUSPENDERS)))
[2] (FA (X) (IF (IN X FIRECHIEF) (IN X FIREMEN)))

are universally quantified expressions of the following rules:

[1] IF: X IS A FIREMAN
THEN: X OWNS RED SUSPENDERS

[2] IF: X IS A FIRECHIEF
THEN: X IS A FIREMAN

Green's program uses such "rules" to answer questions regarding system knowledge. The questions themselves may be stated as rules:

[3] Question: (FA (X) (IF
(IN X FIRECHIEF) (OWNS X RED-SUSPENDERS)))

that is,

[3] Is the following rule valid?
IF: X IS A FIRECHIEF
THEN: X OWNS RED SUSPENDERS

QA3 uses [1] and [2], plus the "resolution principle" for theorem proving [Robinson, 1965], to show that [3] is a valid rule and thereby to answer the question affirmatively. Resolution is mentioned again during the discussion of control structures in § 3.3.1.

The second example of a rule-based system is the program designed by Colby *et al.* for modeling human belief structures [Colby, 1969]. They acquired statements of belief from a human subject and coded them as either facts or rules of inference. Facts had associated numerical weights representing their degree of credibility to the human subject, but the rules reflected simple implication without any weighting of the strength of the relationship. For example:

(F 80 SELF NOTLIKE (CHILD1 HAS AGGRESSIVENESS))

is their system's representation for the fact (F) that the subject (SELF) found it strongly credible (80) that she did not like the

aggressiveness of one of her children (CHILD1). A sample rule from their data base is:

(R THEPARENT SLAP HISCHILD IMPLIES THEPARENT DISTRESS HISCHILD)

"Implies" in their rules does not necessarily correspond to logical implication. Instead it may represent relationships that are logical, causal, temporal, or conceptual. Furthermore, the rules are similar to those of MYCIN in that they represent judgments of a human subject (*cf.* expert) rather than natural laws.

The main task for Colby and his coworkers involved estimating the credibility of a given proposition describing some actual or hypothetical situation. They tested their model by writing a program that used the belief structures obtained from their human subject in order to assess the credibility of a new hypothesis not already in the data base. They then compared the judgment of the program with the credibility estimate of the subject herself. System rules and facts were linked in a graph structure that was searched by a variety of algorithms in an attempt to assess the credibility of a new proposition. Unfortunately, the human subject left the study before a formal evaluation of the program's credibility estimates could be undertaken.

In the late 1960's, Waterman developed a rule-based system for playing poker [Waterman, 1970]. He selected this game because, unlike chess or other games commonly modeled by computer programs, poker is characterized by imperfect knowledge regarding the opponent's position. Close attention was paid to the optimal representation of heuristics needed by a poker playing machine. He decided that a good representation should:

(1) permit separation of the heuristics from the main body of the program;
(2) provide identification of individual heuristics and an indication of how they are interrelated;
(3) be compatible with generalization schemes.

Clearly these desiderata correspond closely to the criterion of knowledge modularity I discussed in Chapter 2. Waterman's concern with these factors stemmed from his desire to create a program that would

not only play poker but also learn new heuristics that could be incorporated in a straightforward fashion and would permit improvement of the system's game over time.

Waterman pictured poker as a succession of states, with each play causing a transition from one state to another. The situation at any given time could therefore be characterized by a state vector, and game heuristics would involve decisions based upon the current status of the state vector. Thus heuristics could be represented as production rules or so-called situation-action (SA) rules, i.e., if S is true, then take action A. I shall not present Waterman's formal representation here since that would necessitate a description of his rather complex state vector, but the following excerpt from his paper [Waterman, 1970] should give an adequate description of the kind of heuristic rules that he was able to code:

> If your hand is excellent then bet low if the opponent tends to be a conservative player and has just bet low. Bet high if the opponent is not conservative, is not easily bluffed, and has just made a sizable bet. Call if the pot is extremely large, and the opponent has just made a sizable bet.

The program could be taught such heuristics explicitly and was also able to generalize new rules from its experience when playing the game. The result was a system that eventually played an admirable game of poker.

The last rule-based system for discussion in this section is one of the foremost examples of AI techniques effectively applied to a real-world problem domain. HEURISTIC DENDRAL is a large set of programs designed to aid in the identification of chemical structures from mass spectral data [Feigenbaum, 1968; Buchanan, 1969]. The input to the system is the data derived for an unknown organic molecule that has been subjected to mass spectral analysis. HEURISTIC DENDRAL uses this input, plus a complex theory of mass spectroscopy embodied in SA rules, to suggest one or more topological structural formulae for the unknown molecule. The program has a heuristic hypothesis generator that first compiles a set of all reasonable structures on the basis of primary spectral observations. It then uses SA rules acquired from experts in mass spectroscopy to predict spectra for each of the structural hypotheses. A final evaluation stage selects the one or more hypotheses for which the predicted

mass spectrum most closely resembles the spectrum that was empirically observed.

Acquiring the mass spectral rules from experts in organic chemistry, who may have limited knowledge of computers or of the DENDRAL program, has proven to be a task of considerable difficulty [Buchanan, 1970]. One is immediately reminded of the challenge in getting well-formed decision rules for MYCIN by discussing patients with infectious disease experts. An example of one of DENDRAL's SA rules is the following:

```
                1   2 3   4   5   6   7
SA Rule: (C,H)-CH2-NH-CH2-CH2-CH2-(C,H) → Breakbond (4 5)
```

This rule states that a seven membered chain with the characteristics shown in the Situation part of the rule is apt to undergo a bond break between atoms 4 and 5 when subjected to mass spectral bombardment. It is therefore useful in predicting the spectrum of a molecule that satisfies the situation part of the rule (since peaks in a mass spectrum correspond to molecular fragments of a specific identifiable mass).

The SA rules used by HEURISTIC DENDRAL have many similarities to those used in Waterman's program [Waterman, 1970]. Just as Waterman chose a production rule system in part so that new heuristics could be learned and integrated with ease, DENDRAL has broadened its scope to consider mechanisms for inferring new SA rules. This adjunct to HEURISTIC DENDRAL is known as META-DENDRAL [Buchanan, 1971, 1972]. The idea is to analyze the spectra of known molecules in an effort to infer the theoretical basis for the data that are observed. Because system knowledge is maintained in modular SA rules and is not embedded within the programs themselves, this kind of system enhancement is greatly facilitated. The result is a program that often performs at the level of a post-doctoral chemist and is able to analyze and draw inferences on such complex cyclic structures as estrogenic steroids [Buchanan, 1973].

The decision criteria stored in MYCIN's rules are in many ways similar to the "rules" or "theorems" that form the knowledge base of the programs I have discussed. All the systems keep their rules separate from their programs so that the functions are domain independent and attempts at generalization are facilitated. As dis-

cussed in § 3.3.1, the rules are actually used in a variety of fashions. Regardless of control structures, however, the advantages of identifiable packets of knowledge should now be clear. A final point to note is that, unlike the rules in the other systems described, MYCIN's decision criteria contain explicit weighting factors that reflect the strength of the indicated inference.

3.2.1-2 Representation of Rules

The 200 rules currently in the MYCIN system consist of a PREMISE, an ACTION, and sometimes an ELSE clause. Every rule has a name of the form "RULE###" where "###" is a three digit number. When discussing rules in their most general form, it will often be useful to adopt a shortened form of notation. I shall use upper-case letters for conditions and conclusions, inserting a right arrow to indicate implication. Thus

A & B → C

signifies the rule for which the PREMISE is the conjunction of conditions A and B and the ACTION is C.

The details of rules and how they are used are discussed throughout the remainder of this chapter. I therefore offer a formal definition of rules that will serve in part as a guide for what is to follow. The rules are stored as LISP data structures in accordance with the following Backus Normal Form (BNF) description:

```
      <rule> : : = <premise> <action> | <premise> <action> <else>
   <premise> : : = ($AND <condition> . . . <condition>)
 <condition> : : = ( <func1> <context> <parameter> ) |
                   ( <func2> <context> <parameter> <value> ) |
                   ( <special-func> <arguments> ) |
                   ($OR <condition> . . . <condition> )
    <action> : : = <concpart>
      <else> : : = <concpart>
  <concpart> : : = <conclusion> | <actfunc> |
                   (DO-ALL <conclusion> . . . <conclusion> ) |
                   (DO-ALL <actfunc> . . . <actfunc> )
   <context> : : = See § 3.2.2
 <parameter> : : = See § 3.2.3
     <value> : : = See § 3.2.3
```

<func1> : : = See § 3.2.5
<func2> : : = See § 3.2.5
<special-func> : : = See § 3.2.6-2
<arguments> : : = See § 3.2.6-2
<conclusion> : : = See § 3.3.3-2
<actfunc> : : = See § 3.5

Thus the PREMISE of a rule consists of a conjunction of conditions, each of which must hold for the indicated ACTION to be taken. Negations of conditions are handled by the individual predicates (<func1> and <func2>) and therefore do not require a $NOT function to complement the Boolean functions $AND and $OR. If the PREMISE of a rule is known to be false, the conclusion or action indicated by the ELSE clause is taken. If the truth of the PREMISE cannot be ascertained, or the PREMISE is false but no ELSE condition exists, the rule is simply ignored.

The PREMISE of a rule is always a conjunction of one or more conditions. Disjunctions of conditions may be represented as multiple rules with identical ACTION clauses. A condition, however, may itself be a disjunction of conditions. These conventions are somewhat arbitrary but do provide sufficient flexibility so that any Boolean expression may be represented by one or more rules. As is discussed in § 3.3, multiple rules are effectively OR'ed together by MYCIN's control structure.

For example, 2-leveled Boolean nestings of conditions are acceptable as follows:

Legal:

[1] A & B & C → D
[2] A & (B OR C) → D
[3] (A or B or C) & (D or E) → F

Illegal:

[4] A or B or C → D
[5] A & (B or (C & D)) → E

Rule [4] is correctly represented by the following three rules:

[6] A → D
[7] B → D
[8] C → D

whereas [5] must be written as:

[9] A & C & D → E
[10] A & B → E

Unlike rules that involve strict implication, the strength of an inference in MYCIN's rules may be modified by a certainty factor (CF). A CF is a number from −1 to +1, the nature of which is described in § 3.2.4 and in Chapter 4. The notation for indicating the strength of an implication will be as follows:

A & B & C −a→ D

Here the rule states that the conjunction of conditions A, B, and C implies D with certainty factor a.

The following three examples are rules from MYCIN that have been translated into English from their internal LISP representation (§ 3.2.7). They represent the range of rule types available to the system. The details of their internal representation will be explained as I proceed.

RULE037

IF: 1) THE IDENTITY OF THE ORGANISM IS NOT KNOWN WITH CERTAINTY, AND
2) THE STAIN OF THE ORGANISM IS GRAMNEG, AND
3) THE MORPHOLOGY OF THE ORGANISM IS ROD, AND
4) THE AEROBICITY OF THE ORGANISM IS AEROBIC
THEN: THERE IS STRONGLY SUGGESTIVE EVIDENCE (.8) THAT THE CLASS OF THE ORGANISM IS ENTEROBACTERIACEAE

RULE145

IF: 1) THE THERAPY UNDER CONSIDERATION IS ONE OF: CEPHALOTHIN CLINDAMYCIN ERYTHROMYCIN LINCOMYCIN VANCOMYCIN, AND
2) MENINGITIS IS AN INFECTIOUS DISEASE DIAGNOSIS FOR THE PATIENT
THEN: IT IS DEFINITE (1) THE THE THERAPY UNDER CONSIDERATION IS NOT A POTENTIAL THERAPY FOR USE AGAINST THE ORGANISM

RULE060

IF: THE IDENTITY OF THE ORGANISM IS BACTEROIDES
THEN: I RECOMMEND THERAPY CHOSEN FROM AMONG THE FOLLOWING DRUGS:

1 - CLINDAMYCIN	(.99)
2 - CHLORAMPHENICOL	(.99)
3 - ERYTHROMYCIN	(.57)
4 - TETRACYCLINE	(.28)
5 - CARBENICILLIN	(.27)

Before I can explain how rules such as these are invoked and evaluated, it is necessary further to describe MYCIN's internal organization. I shall therefore temporarily digress in order to lay some groundwork for the description of the evaluation functions in § 3.2.5.

3.2.2 CATEGORIZATION OF RULES BY CONTEXT

3.2.2-1 Context Tree

Although it is common to describe diagnosis as inference based on attributes of the patient, MYCIN's decisions must necessarily involve not only the patient but also the cultures that have been grown, organisms isolated, and drugs that have been administered. Each of these is termed a "context" of the program's reasoning (see <context> in the BNF description of rules, § 3.2.1-2). (This use of the word "context" should not be confused with its meaning in high level languages that permit temporary saving of all information regarding a program's current status–a common mechanism for backtracking and parallel processing implementations).

MYCIN currently knows about ten different context-types:

CURCULS	- a current culture from which organisms were isolated
CURDRUGS	- an antimicrobial agent currently being administered to a patient
CURORGS	- an organism isolated from a current culture
OPDRGS	- an antimicrobial agent administered to the patient during a recent operative procedure
OPERS	- an operative procedure which the patient has undergone
PERSON	- the patient himself

POSSTHER - a therapy being considered for recommendation
PRIORCULS - a culture obtained in the past
PRIORDRGS - an antimicrobial agent administered to the patient previously
PRIORORGS - an organism isolated from a prior culture

Except for PERSON, each of these context-types may be instantiated more than once during any given run of the consultation program. Some may not be created at all if they do not apply to the given patient. However, each time a context-type is instantiated it is given a unique name. For example, CULTURE-1 is the first CURCUL and ORGANISM-1 is the first CURORG. Subsequent CURCULS or PRIORCULS are called CULTURE-2, CULTURE-3, etc.

The context-types instantiated during a run of the consultation program are arranged hierarchically in a data structure termed the "context tree." One such tree is shown in Figure 3-1. The context-type for each instantiated context is shown in parentheses beside its name. Thus, to clarify terminology, we note that a node in the context tree is called a context and is created as an instantiation of a

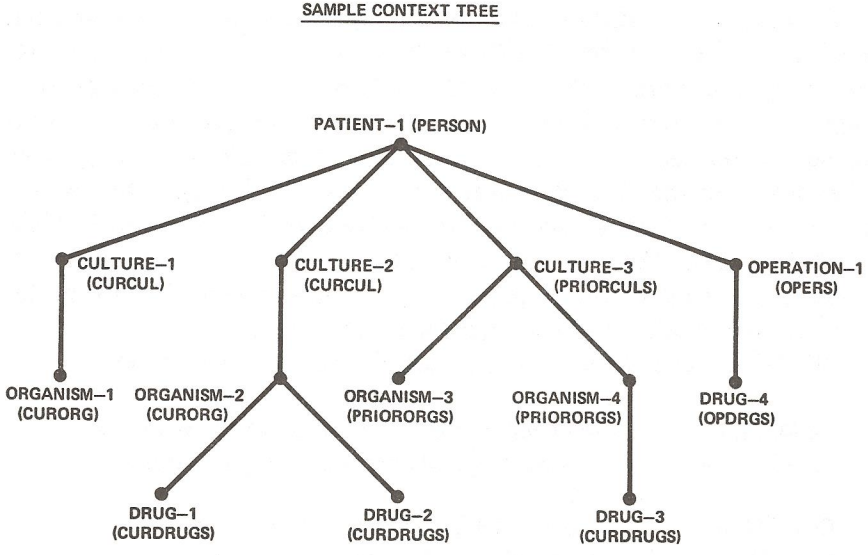


Figure 3-1: Context tree for a sample patient with two recent positive cultures, an older one, and a recent significant operative procedure. Nodes in the tree are termed "contexts."

context-type. This sample context tree corresponds to a patient from whom two current cultures and one prior culture were obtained. One organism was isolated from each of the current cultures, but the patient is being treated (with two drugs) for only one of the current organisms. Furthermore, two organisms were grown from the prior culture but therapy was instituted to combat only one of these. Finally, the patient has had a recent operative procedure during which he was treated with an antimicrobial agent.

The context tree is useful not only because it gives structure to the clinical problem (Figure 3-1 already tells us a good deal about PATIENT-1), but also because we often need to be able to relate one context to another. For example, in considering the significance of ORGANISM-2, MYCIN may well want to be able to reference the site of the culture from which ORGANISM-2 was obtained. Since the patient has had three different cultures, we need an explicit mechanism for recognizing that ORGANISM-2 came from CULTURE-2, not CULTURE-1 or CULTURE-3. The technique for dynamic propagation (i.e., growth) of the context tree during a consultation is described in § 3.4).

3.2.2-2 Interrelationship of Rules and Context Tree

The 200 rules currently used by MYCIN are not explicitly linked in a decision tree or reasoning network. This feature is in keeping with our desire to keep the system knowledge modular and manipulable. However, rules are subject to categorization in accordance with the context-types for which they are most appropriately invoked. For example, some rules deal with organisms, some with cultures, and still others deal solely with the patient himself. MYCIN's current rule categories are as follows (context-types to which they may be applied are enclosed in parentheses):

CULRULES	-	rules that may be applied to any culture (CURCULS or PRIORCULS)
CURCULRULES	-	rules that may be applied only to current cultures (CURCULS)
CURORGRULES	-	rules that may be applied only to current organisms (CURORGS)

DRGRULES	-	rules that may be applied to any antimicrobial agent that has been administered to combat a specific organism (CURDRUGS PRIORDRGS)
OPRULES	-	rules that may be applied to operative procedures (OPERS)
ORDERRULES	-	rules that are used to order the list of possible therapeutic recommendations (POSSTHER)
ORGRULES	-	rules that may be applied to any organism (CURORGS or PRIORORGS)
PATRULES	-	rules that may be applied to the patient (PERSON)
PDRGRULES	-	rules that may be applied only to drugs given to combat prior organisms (PRIORDRUGS)
PRCULRULES	-	rules that may be applied only to prior cultures (PRIORCULS)
PRORGRULES	-	rules that may be applied only to organism isolated from prior cultures (PRIORORGS)
THERULES	-	rules that store information regarding drugs of choice (§ 3.5).

Every rule in the MYCIN system belongs to one, and only one, of these categories. Furthermore, selecting the proper category for a newly acquired rule does not present a problem. In fact, as is discussed in § 6.3, category selection can be automated to a large extent.

Consider now a rule such as:

RULE124

IF: 1) THE SITE OF THE CULTURE IS THROAT, AND
2) THE IDENTITY OF THE ORGANISM IS STREPTOCOCCUS
THEN: THERE IS STRONGLY SUGGESTIVE EVIDENCE (.8) THAT THE SUBTYPE OF THE ORGANISM IS NOT GROUP-D

This is one of MYCIN's ORGRULES and may thus be applied either to a CURORGS context or a PRIORORGS context. Referring back to Figure 3-1, suppose RULE124 above were applied to ORGANISM-2. The first condition in the PREMISE refers to the site of the culture from which ORGANISM-2 was isolated (i.e., CULTURE-2) and not to the organism itself (i.e., organisms do not have SITES, but cultures do). The context tree is therefore impor-

tant, as I mentioned above, for determining the proper context when a rule refers to an attribute of a node in the tree other than the context to which the rule is being explicitly applied. Note that this means that a single rule may refer to nodes at several levels in the context tree. The rule is categorized simply on the basis of the lowest context-type (in the tree) that it may reference. Thus RULE124 is an ORGRULE rather than a CULRULE.

3.2.3 CLINICAL PARAMETERS

This subsection describes the data types indicated by <parameter> and <value> in the BNF description of rules (§ 3.2.1-2). Although I have previously asserted that all MYCIN's knowledge is stored in its corpus of rules, the clinical parameters and their associated properties comprise an important class of second level knowledge. I shall first explain the kind of parameters used by the system, and will then describe their representation.

A clinical parameter is a characteristic of one of the contexts in the context tree, i.e., the name of the patient, the site of a culture, the morphology of an organism, the dose of a drug, etc. All such attributes will be termed "clinical parameters." A patient's status would be completely specified by a context tree in which values were known for all the clinical parameters characterizing each node in the tree (assuming the parameters known to MYCIN encompass all those that are clinically relevant—a dubious assumption at present). In general this is more information than is needed, however, so one of MYCIN's tasks is to identify those clinical parameters that need to be considered for the patient about whom advice is being sought. This task is similar to the problem of sequential test selection that was relevant to many of the programs discussed in § 1.3.

The concept of an attribute-object-value triple is common to much of the AI field. This associative relationship is a basic data type for the SAIL language [Feldman, 1972] and is the foundation for the property-list formalism in LISP [McCarthy, 1962]. Relational predicates in the predicate calculus also represent associative triples. The point is that many facts may be expressed as triples that state that some object has an attribute with some specified value. Stated in the order <attribute object value>, examples include:

(COLOR BALL RED)
(OWNS FIREMAN RED-SUSPENDERS)
(AGE BOB 22)
(FATHER CHILD "DADDY")
(GRAMSTAIN ORGANISM GRAM-POSITIVE)
(DOSE DRUG 1.5-GRAMS)
(MAN BOB TRUE)
(WOMAN BOB FALSE)

Note that the last two examples are different from the others since they represent a rather different kind of relationship. In fact, several authors would classify the first six as "relations" and the last two as "predicates," using the simpler notation:

MAN(BOB)
–WOMAN(BOB)

Regardless of whether it is written as MAN(BOB) or (MAN BOB TRUE), this binary predicate statement has rather different characteristics from the relations that form natural triples. This distinction will become more clear later (see "yes–no" parameters below).

MYCIN stores inferences and data using the attribute-object-value concept I have just described. The object is always some context in the context tree, and the attribute is a clinical parameter appropriate for that context. Information stored using this mechanism may be retrieved and updated in accordance with a variety of conventions described throughout this chapter.

3.2.3-1 Three Kinds of Clinical Parameters

There are three fundamentally different kinds of clinical parameters. The simplest variety are the ones we call "single-valued" parameters. These are attributes such as the name of the patient or the identity of the organism. In general, they have a large number of possible values that are mutually exclusive. As a result, only one can be the true value, although several may seem likely at any point during the consultation.

"Multi-valued" parameters also generally have a large number of possible values. The difference is that the possible values need not be

mutually exclusive. Thus, such attributes as a patient's drug allergies or a locus of infection may have multiple values, each of which is known to be correct.

The third kind of clinical parameter corresponds to the binary predicate discussed above. These are attributes that are either true or false for the given context. For example, the significance of an organism is either true or false (yes or no), as is the parameter indicating whether the dose of a drug is adequate. Attributes of this variety are called "yes–no" parameters. They are, in effect, a special kind of "single-valued" parameter for which there are only two possible values.

3.2.3-2 Classification and Representation of Parameters

The clinical parameters known to MYCIN are categorized in accordance with the context to which they apply. These categories include:

- PROP-CUL - those clinical parameters that are attributes of cultures (e.g., site of the culture, method of collection)
- PROP-DRG - those clinical parameters that are attributes of administered drugs (e.g., name of the drug, duration of administration)
- PROP-OP - those clinical parameters that are attributes of operative procedures (e.g., the cavity, if any, opened during the procedure)
- PROP-ORG - those clinical parameters that are attributes of organisms (e.g., identity, gram stain, morphology)
- PROP-PT - those clinical parameters that are attributes of the patient (e.g., name, sex, age, allergies, diagnoses)
- PROP-THER - those clinical parameters that are attributes of therapies being considered for recommendation (e.g., recommended dosage, prescribing name)

These categories encompass all clinical parameters used by the system. Note that any of the nodes (contexts) in the context tree for the patient may be fully characterized by the values of the set of clinical parameters in one of these categories.

Each of the 65 clinical parameters currently known to MYCIN has an associated set of properties that is used during consideration of

yes–no parameter

FEBRILE: <FEBRILE is an attribute of a patient and is therefore a member of the list PROP-PT>
EXPECT: (YN)
LOOKAHEAD: (RULE149 RULE109 RULE045)
PROMPT: (Is * febrile?)
TRANS: (* IS FEBRILE)

single-valued parameter

IDENT: <IDENT is an attribute of an organism and is therefore a member of the list PROP-ORG>
CONTAINED-IN: (RULE030)
EXPECT: (ONEOF (ORGANISMS)
LABDATA: T
LOOKAHEAD: (RULE004 RULE054 . . . RULE168)
PROMPT: (Enter the identity (genus) of * :)
TRANS: (THE IDENTITY OF *)
UPDATED-BY: (RULE021 RULE003 . . . RULE166)

multi-valued parameter

INFECT: <INFECT is an attribute of a patient and is therefore a member of the list PROP-PT>
EXPECT: (ONEOF PERITONITIS BRAIN-ABSCESS MENINGITIS BACTEREMIA UPPER-URINARY-TRACT-INFECTION . . . ENDOCARDITIS)
LOOKAHEAD: (RULE115 RULE149 . . . RULE045)
PROMPT1: (Is there evidence that the patient has a (VALU) ?)
TRANS: (AN INFECTIOUS DISEASE DIAGNOSIS FOR *)
UPDATED-BY: (RULE157 RULE022 . . . RULE105)

Figure 3-2: Examples of the three types of clinical parameters. As shown, each clinical parameter is characterized by a set of "properties" described in the text.

the parameter for a given context. Figure 3-2 presents three clinical parameters that together demonstrate several of these properties:

EXPECT - this property indicates the range of expected values that the parameter may have.
if = (YN) then the parameter is a "yes–no" parameter
if = (NUMB) then the expected value of the parameter is a number

if = (ONEOF <list>) then the value of the parameter must be a member of <list>

if = (ANY) then there is no restriction on the range of values that the parameter may have

PROMPT - this property is a sentence used by MYCIN when it requests the value of the clinical parameter from the user; if there is an asterisk in the phrase (see Figure 3-2), it is replaced by the name of the context about which the question is being asked; this property is used only for "yes–no" or "single-valued" parameters.

PROMPT1 - this property is similar to PROMPT except it is used if the clinical parameter is a "multi-valued" parameter; in these cases MYCIN only asks the question about a single one of the possible parameter values; the value of interest is substituted for (VALU) in the question.

LABDATA - this property is a flag that is either T or NIL; if T it indicates that the clinical parameter is a piece of primitive data, the value of which may be known with certainty to the user (see § 3.3.2-1).

LOOKAHEAD - this property is a list of all rules in the system that reference the clinical parameter in their PREMISE.

UPDATED-BY - this property is a list of all rules in the system in which the ACTION or ELSE clause permits a conclusion to be made regarding the value of the clinical parameter.

CONTAINED-IN - this property is a list of all rules in the system in which the ACTION or ELSE clause references the clinical parameter but does not cause its value to be updated.

TRANS - this property is used for translating the clinical parameter into its English representation (see § 3.2.7); the context of the parameter is substituted for the asterisk during translation.

DEFAULT - this property is used only with clinical parameters for which EXPECT = (NUMB); it gives the expected units for numerical answers (e.g., days, years, grams, etc.)

CONDITION - this property, when utilized, is an executable LISP expression that is evaluated before MYCIN requests the value of the parameter; if the CONDITION is true, the question is not asked (e.g., "Don't ask for an organism's subtype if its genus is not known by the user").

The uses of these properties will be discussed throughout the remainder of this chapter and in Chapter 5. However, a few additional points are relevant here. First, it should be noted that the order of

rules on the properties LOOKAHEAD, UPDATED-BY, and CONTAINED-IN is arbitrary and does not affect the program's advice. Second, TRANS is the only property that must exist for every clinical parameter. Thus, for example, if there is no PROMPT or PROMPT1 stored for a parameter, the system assumes that it simply cannot ask the user for the value of the parameter. Finally, note in Figure 3-2 the difference in the TRANS property for "yes–no" and non-"yes–no" parameters. In general, a parameter and its value may be translated as:

THE <attribute> OF <object> IS <value>

However, for a "yes–no" parameter such a FEBRILE, it is clearly necessary to translate the parameter in a fashion other than:

THE FEBRILE OF PATIENT-1 IS YES

Our solution has been to suppress the YES altogether and simply to say:

PATIENT-1 IS FEBRILE

3.2.4 CERTAINTY FACTORS

Chapter 4 presents a detailed description of certainty factors and their theoretical foundation. This section therefore provides only a brief overview of the subject. A familiarity with the characteristics of certainty factors (CF's) is necessary, however, for the discussion of MYCIN during the remainder of this chapter.

The value of every clinical parameter is stored by MYCIN along with an associated certainty factor that reflects the system's "belief" that the value is correct. This formalism is necessary because, unlike domains in which objects either have or do not have some attribute, in medical diagnosis and treatment there is often uncertainty regarding attributes such as the significance of the disease, the efficacy of a treatment, or the diagnosis itself. As discussed in § 1.3, most medical decision making programs use probability to reflect the uncertainties. CF's are an alternative to conditional probability that offer several advantages in MYCIN's domain (as described in Chapter 4).

A certainty factor is a number between −1 and +1 that reflects the

degree of belief in a hypothesis. Positive CF's indicate there is evidence that the hypothesis is valid. The larger the CF, the greater the belief in the hypothesis. When CF=1, the hypothesis is known to be correct. On the other hand, negative CF's indicate that the weight of evidence suggests that the hypothesis is false. The smaller the CF, the greater the belief that the hypothesis is invalid. CF=−1 means that the hypothesis has been effectively disproven. When CF=0, there is either no evidence regarding the hypothesis, or the supporting evidence is equally balanced by evidence suggesting that the hypothesis is not true.

MYCIN's hypotheses are statements regarding values of clinical parameters for the various nodes in the context tree. For example, sample hypotheses are:

h1 = The identity of ORGANISM-1 is streptococcus
h2 = PATIENT-1 is febrile
h3 = The name of PATIENT-1 is John Jones

We use the notation CF[h,E]=X to represent the certainty factor for the hypothesis h based upon evidence E. Thus if CF[h1,E]=.8, CF[h2,E]=−.3, and CF[h3,E]=+1, the three sample hypotheses above may be qualified as follows:

CF[h1,E]=.8	:	There is strongly suggestive evidence (.8) that the identity of ORGANISM-1 is streptococcus
CF[h2,E]=−.3	:	There is weakly suggestive evidence (.3) that PATIENT-1 is not febrile
CF[h3,E]=+1	:	It is definite (1) that the name of PATIENT-1 is John Jones

Certainty factors are used in two ways. First, as noted, the value of every clinical parameter is stored with its associated certainty factor. In this case the evidence E stands for all information currently available to MYCIN. Thus, if the program needs the identity of ORGANISM-1, it may look in its dynamic data base and find:

IDENT of ORGANISM-1 = ((STREPTOCOCCUS .8))

The second use of CF's is in the statement of decision rules themselves. In this case the evidence E corresponds to the conditions in the PREMISE of the rule. Thus

A & B & C –x→D

is a representation of the statement CF[D, (A&B&C)] =x. For example, consider the following rule:

IF: 1) THE STAIN OF THE ORGANISM IS GRAMPOS, AND
2) THE MORPHOLOGY OF THE ORGANISM IS COCCUS, AND
3) THE GROWTH CONFORMATION OF THE ORGANISM IS CHAINS

THEN: THERE IS SUGGESTIVE EVIDENCE (.7) THAT THE IDENTITY OF THE ORGANISM IS STREPTOCOCCUS

This rule may also be represented as CF[h1,e] =.7 where h1 is the hypothesis that the organism (context of the rule) is a streptococcus and e is the evidence that it is a gram positive coccus growing in chains.

Since diagnosis is, in effect, the problem of selecting a disease from a list of competing hypotheses, it should be clear that MYCIN may simultaneously be considering several hypotheses regarding the value of a clinical parameter. These hypotheses are stored together, along with their CF's, for each node in the context tree. We use the notation Val[C,P] to signify the set of all hypotheses regarding the value of the clinical parameter P for the context C. Thus if MYCIN has reason to believe that ORGANISM-1 may be either a streptococcus or staphylococcus, although pneumococcus has been ruled out, its dynamic data base might well show:

```
Val[ORGANISM-1,IDENT] = ((STREPTOCOCCUS .6)
                         (STAPHYLOCOCCUS .4)
                         (DIPLOCOCCUS-PNEUMONIAE –1))
```

Note that Chapter 5 shows that the sum of the CF's for supported hypotheses regarding a "single-valued" parameter (i.e., those parameters for which the hypotheses are mutually exclusive) should not exceed 1. "Multi-valued" parameters, on the other hand, may have several hypotheses that are all known to be true. For example:

```
Val[PATIENT-1,ALLERGY] = ((PENICILLIN 1) (AMPICILLIN 1)
                          (CARBENICILLIN 1) (METHICILLIN 1))
```

As soon as a hypothesis regarding a "single-valued" parameter is

proved to be true, all competing hypotheses are effectively disproved:

```
Val[ORGANISM-1,IDENT] = ((STREPTOCOCCUS 1)
                         (STAPHYLOCOCCUS -1)
                         (DIPLOCOCCUS-PNEUMONIAE -1))
```

In Chapter 4 it is demonstrated that CF[h,E] = −CF[not.h,E]. This observation has important implications for the way MYCIN handles the binary-valued attributes we call "yes–no" parameters. Since "yes" is "not.no," it is not necessary to consider "yes" and "no" as competing hypotheses for the value of a "yes–no" parameter (as we do for "single-valued" parameters). Instead we can always express "no" as "yes" with a reversal in the sign of the CF. This means that Val[C,P] is always equal to the single value "yes," along with its associated CF, when P is a "yes–no" parameter.

In § 3.3.3-2, I discuss MYCIN's mechanism for adding to the list of hypotheses in Val[C,P] as new rules are invoked and executed. The following points should be emphasized here, however:

(1) the strength of the conclusion associated with the execution of a rule reflects not only the CF assigned to the rule, but also the program's degree of belief regarding the validity of the PREMISE;
(2) the support of several rules favoring a single hypothesis may be assimilated incrementally on the list Val[C,P] by using special combining functions described in § 4.6.

3.2.5 FUNCTIONS FOR EVALUATION OF PREMISE CONDITIONS

This section describes the evaluation of the individual conditions (see <condition>, § 3.2.1-2) in the PREMISE of rules. Conditions in general evaluate to "true" or "false" (T or NIL). Thus, they may at first glance be considered simple predicates on the values of clinical parameters. However, since there may be several competing hypotheses on the list Val[C,P], each associated with its own degree of belief as reflected by the CF, conditional statements regarding the value of parameters can be quite complex. All predicates are implemented as LISP functions. The functions that undertake the required analysis are of three varieties, specified by the designations <func1>,

<func2>, and <special-func> in the BNF rule description (§ 3.2.2-1). This section explains the <func1> and <func2> predicates. The <special-func> category is deferred until § 3.2.6-2, however, so that I may first introduce our specialized knowledge structures (§ 3.2.6-1).

There are four predicates in the category <func1>. These functions do not form conditionals on specific values of a clinical parameter, but are concerned with the more general status of knowledge regarding the attributes in question. For example, KNOWN[ORGANISM-1, IDENT] is an invocation of the <func1> predicate KNOWN; it would return true if the identity of ORGANISM-1 were known, regardless of the value of the clinical parameter IDENT. KNOWN and the other <func1> predicates may be formally defined as follows:

Predicates of the class <func1>:

Let V=Val[C,P] be the set of all hypotheses regarding the value of the clinical parameter P for the context C

Let Mv=Max[V] be the most strongly supported hypothesis in V (i.e., the hypothesis with the largest CF)

Let CFmv=CF[Mv,E] be the certainty factor of Mv given E, where E is the total available evidence

Then, if P is either a "single-valued" or "multi-valued" parameter, the four predicates (functions) may be specified as follows:

FUNCTION	IF:	THEN:	ELSE:
KNOWN[C,P]	CFmv>.2	T	NIL
NOTKNOWN[C,P]	CFmv≤.2	T	NIL
DEFINITE[C,P]	CFmv=1	T	NIL
NOTDEFINITE[C,P]	CFmv<1	T	NIL

In words, these definitions reflect MYCIN's convention that the value of a parameter is KNOWN if the CF of the most highly supported hypothesis exceeds .2. The .2 threshold was selected empirically. The implication is that a positive CF less than .2 reflects so little evidence supporting the hypothesis that there is virtually no reasonable hypothesis currently known. The interrelationships among these functions are diagrammed on a CF number line in

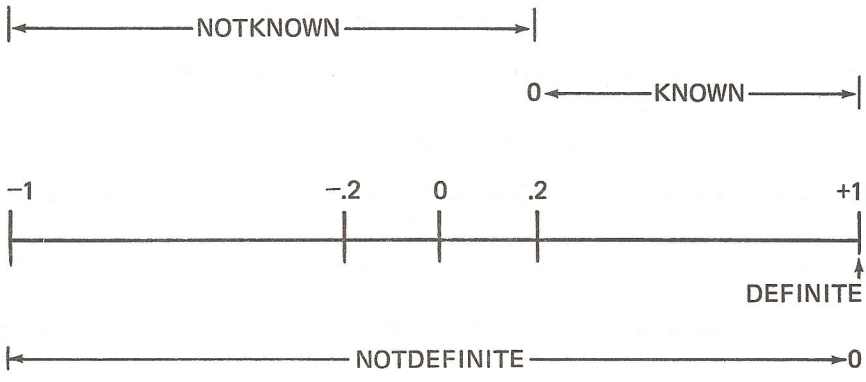


Figure 3-3: Diagram indicating the range of CF values over which the <func1> predicates hold true when applied to multivalued or single-valued (i.e., non-"yes–no") clinical parameters. Vertical lines and zeros distinguish closed and nonclosed certainty factor ranges, respectively.

Figure 3-3. Regions specified are the range of values for CFmv over which the function returns T.

As was pointed out in the previous section, however, "yes–no" parameters are special cases because we know CF[YES,E] = –CF[NO,E]. Since the values of "yes–no" parameters are always stored in terms of YES, MYCIN must recognize that a YES with CF=–.9 is equivalent to a NO with CF=.9. The definitions of our four <func1> predicates above do not reflect this distinction. Therefore, when P is a "yes–no" parameter, the four functions are specified as follows:

FUNCTION:	IF:	THEN:	ELSE:
KNOWN[C,P]	|CFmv|>.2	T	NIL
NOTKNOWN[C,P]	|CFmv|≤.2	T	NIL
DEFINITE[C,P]	|CFmv|=1	T	NIL
NOTDEFINITE[C,P]	|CFmv|<1	T	NIL

Figure 3-4 shows the relationship among these functions for "yes–no" parameters.

There are nine predicates in the category <func2>. Unlike the <func1> predicates, these functions control conditional statements regarding specific values of the clinical parameter in question. For example, SAME[ORGANISM-1, IDENT, E.COLI] is an invocation

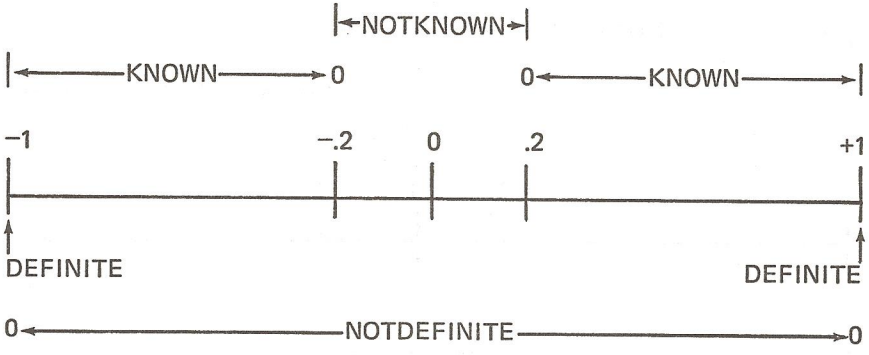


Figure 3-4: Diagram indicating the range of CF values over which the <func1> predicates hold true when applied to "yes–no" clinical parameters.

of the <func2> predicate SAME; it would return true if the identity of ORGANISM-1 were known to be E.coli. SAME and the other <func2> predicates may be formally defined as follows:

Predicates of the class <func2>:

Let V=Val[C,P] be the set of all hypotheses regarding the value of the clinical parameter P for the context C.

Let I=Intersection[V,LST] be the set of all hypotheses in V which also occur in the set LST; LST contains the possible values of P for comparison by the predicate-function; it usually contains only a single element; if no element in LST is also in V, I is simply the empty set.

Let Mi=Max[I] be the most strongly confirmed hypothesis in I; thus Mi is NIL if I is the empty set;

Let CFmi=CF[Mi,E] be the certainty factor of Mi given E, where CFmi=0 if Mi is NIL

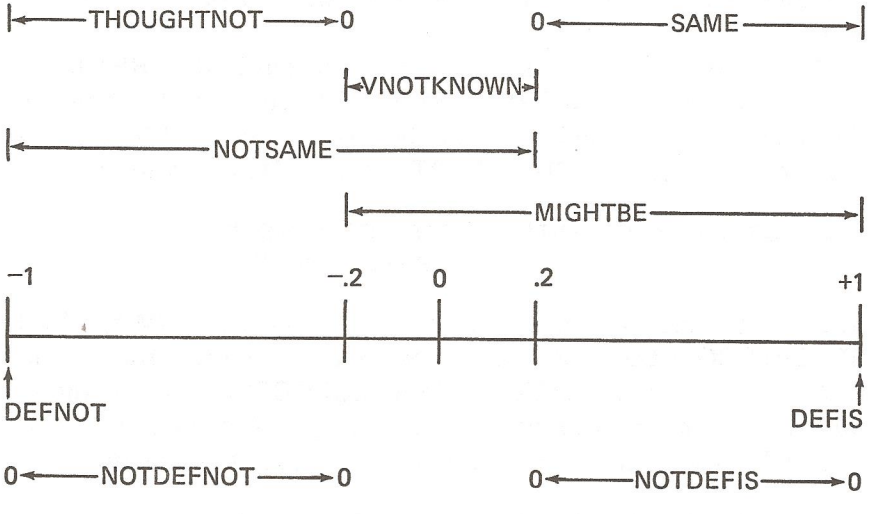
Then the <func2> predicates are defined as follows:

FUNCTION:	IF:	THEN:	ELSE:
SAME[C,P,LST]	CFmi>.2	CFmi	NIL
THOUGHTNOT[C,P,LST]	CFmi<-.2	-CFmi	NIL
NOTSAME[C,P,LST]	CFmi≤.2	T	NIL
MIGHTBE[C,P.LST]	CFmi≥-.2	T	NIL
VNOTKNOWN[C,P,LST]	|CFmi|≤.2	T	NIL
DEFIS[C,P,LST]	CFmi=+1	T	NIL

DEFNOT[C,P,LST]	CFmi=−1	T	NIL
NOTDEFIS[C,P,LST]	.2<CFmi<1	T	NIL
NOTDEFNOT[C,P,LST]	−1<CFmi<−.2	T	NIL

The names of the functions have been selected to reflect their semantics. Figure 3-5 shows a graphic representation of each function and also explicitly states the interrelationships among them.

Note that SAME and THOUGHTNOT are different from all the other functions that I have discussed in that they return a number (CF) rather than T if the defining condition holds. This feature permits MYCIN to record the degree to which PREMISE conditions are satisfied. In order to explain this point, I must discuss the $AND function that oversees the evaluation of the PREMISE of a rule. The reader will recall the BNF description from § 3.2.1-2:



SAME or NOTSAME = THOUGHTNOT or MIGHTBE = T
NOTSAME = VNOTKNOWN or THOUGHTNOT
THOUGHTNOT = NOTDEFNOT or DEFNOT
MIGHTBE = VNOTKNOWN or SAME
SAME = NOTDEFIS or DEFIS

Figure 3-5: Diagram indicating the range of CF values over which the <func2> predicates hold true. The logical relationships of these predicates are also summarized beneath the diagram.

<premise> : : = ($AND <condition> . . . <condition>)

$AND is similar to the standard LISP "AND" function in that it evaluates its conditional arguments one at a time, returning false (NIL) as soon as a condition is found to be false, and otherwise returning true (T). The difference is that $AND expects some of its conditions to return numerical values rather than simply T or NIL. If an argument condition returns NIL (or a number equal to .2 or less) it is considered false and $AND stops considering subsequent arguments. On the other hand, nonnumeric values of conditions are interpreted as indicating truth with CF=1. Thus each true condition either returns a number or a non-NIL value that is interpreted as 1. $AND then maintains a record of the lowest value returned by any of its arguments. This number, termed TALLY, is a certainty tally that indicates MYCIN's degree of belief in the PREMISE (see Combining Function 2 in § 4.6). Thus .2<TALLY≤1, where TALLY=1 indicates that MYCIN believes the PREMISE to be true with certainty.

Most of the predicates that evaluate conditions in the PREMISE of a rule return either T or NIL as we have shown. Consider, however, the semantics of the most commonly used function, SAME, and its analogous function, THOUGHTNOT. Suppose MYCIN knows:

```
Val [ORGANISM-2, IDENT] . = ((STREPTOCOCCUS .7)
                             (STAPHYLOCOCCUS .3))
```

Then it seems clear that SAME[ORGANISM-1,IDENT, STREPTOCOCCUS] is in some sense "more true" than SAME[ORGANISM-1,IDENT,STAPHYLOCOCCUS], even though both hypotheses exceed the threshold CF=.2. If SAME merely returned T, this distinction would be lost. Thus, for this example:

```
SAME[ORGANISM-1,IDENT,STREPTOCOCCUS] = .7
SAME[ORGANISM-1,IDENT,STAPHYLOCOCCUS] = .3
```

whereas

```
KNOWN[ORGANISM-1,IDENT] = T
```

and

```
NOTDEFIS[ORGANISM-1,IDENT,STREPTOCOCCUS] = T
```

A similar argument explains why THOUGHTNOT returns a CF rather than T. It is unclear whether any of the other <func2> predicates should return a CF rather than T; my present conviction is that the semantics of those functions do not require relative weightings in the way that SAME and THOUGHTNOT do.

Let me give a brief example, then, of the way in which the PREMISE of a rule is evaluated by $AND. Consider the following ORGRULE:

```
IF:   1) THE STAIN OF THE ORGANISM IS GRAMNEG, AND
      2) THE MORPHOLOGY OF THE ORGANISM IS ROD, AND
      3) THE AEROBICITY OF THE ORGANISM IS AEROBIC
THEN: THERE IS STRONGLY SUGGESTIVE EVIDENCE (.8) THAT THE
      CLASS OF THE ORGANISM IS ENTEROBACTERIACEAE
```

which is internally coded in LISP as:

```
PREMISE: ($AND  (SAME CNTXT GRAM GRAMNEG)
                (SAME CNTXT MORPH ROD)
                (SAME CNTXT AIR AEROBIC))
ACTION:  (CONCLUDE CNTXT CLASS
         ENTEROBACTERIACEAE TALLY .8)
```

Suppose this rule has been invoked for consideration of ORGANISM-1, i.e., the context of the rule (CNTXT) is the node in the context tree termed ORGANISM-1. Now suppose that MYCIN has the following information in its data base (how it gets there is the subject of § 3.3.3):

```
Val[ORGANISM-1,GRAM] = ((GRAMNEG 1.0))
Val[ORGANISM-1,MORPH] = ((ROD .8) (COCCUS .2))
Val[ORGANISM-1,AIR] = ((AEROBIC .6) (FACUL .4))
```

$AND begins by evaluating SAME[ORGANISM-1,GRAM, GRAMNEG]. The function returns CF=1.0, so TALLY is set to 1.0 (see definition of TALLY in the description of $AND above). Next $AND evaluates the second PREMISE condition, SAME [ORGANISM-1,MORPH,ROD], which returns 0.8. Since the first two conditions both were found to hold, $AND evaluates SAME [ORGANISM-1,AIR,AEROBIC] which returns 0.6. Thus, TALLY is set to 0.6 and $AND returns T. Since the PREMISE is true, MYCIN may now draw the conclusion indicated in the ACTION portion of

the rule. Note, however, that CONCLUDE has as arguments both .8 (i.e., the CF for the rule as provided by the expert) and TALLY (i.e., the certainty tally for the PREMISE). CONCLUDE and the other functions that control inferences are described in § 3.3.3-2.

3.2.6 (*) STATIC KNOWLEDGE STRUCTURES

Although all MYCIN's inferential knowledge is stored in rules, there are various kinds of static definitional information that are stored differently even though they are accessible from rules.

3.2.6-1 (*) Tabular and List-based Knowledge

There are three categories of knowledge structures that could be discussed in this section. However, one of them, MYCIN's 800-word dictionary, is used principally for natural language understanding. Its details are described elsewhere [Shortliffe, 1974b]. The other two data structures are simple linear lists and knowledge tables.

Simple lists: Simple lists provide a mechanism for simplifying references to variables and optimizing knowledge storage by avoiding unnecessary duplication. Two examples should be sufficient to explain this point.

In § 3.2.3-2, I showed that the EXPECT property for the clinical parameter IDENT is:

```
(ONEOF (ORGANISMS))
```

ORGANISMS is the name of a linear list containing the names of all bacteria known to MYCIN (see § 1.5.1). There is also a clinical parameter named COVERFOR for which the EXPECT property is:

```
(ONEOF ENTEROBACTERIACEAE (ORGANISMS) G+COCCI G-COCCI)
```

Thus, by storing the organisms separately on a list named ORGANISMS, we avoid having to duplicate the list of names in the EXPECT property of both IDENT and COVERFOR. Furthermore, using the variable name rather than internal pointers to the list structure facilitates references to the list of organisms whenever it is needed.

A second example involves the several rules in the system that make conclusions based on whether an organism was isolated from a site that is normally sterile or nonsterile. STERILESITES is the name of a simple list containing the names of all normally sterile sites known to the system. There is a similar list named NONSTERILESITES. Thus many rules can have the condition (SAME CNTXT SITE STERILESITES) and the sites need not be listed explicitly in each rule.

Knowledge tables: In conjunction with the special functions discussed in the next subsection, MYCIN's knowledge tables permit a single rule to accomplish a task that would otherwise require several rules. A knowledge table contains a comprehensive record of certain clinical parameters plus the values they take on under various circumstances. For example, one of MYCIN's knowledge tables itemizes the gramstain, morphology, and aerobicity for every bacterial genus known to the system. Consider, then, the task of inferring an organism's gram stain, morphology, and aerobicity if its identity is known with certainty. Without the knowledge table, MYCIN would require several rules of the form:

IF: THE IDENTITY OF THE ORGANISM IS DEFINITELY W
THEN: 1) IT IS DEFINITE (1) THAT THE GRAMSTAIN OF THE ORGANISM IS X, AND
2) IT IS DEFINITE (1) THAT THE MORPHOLOGY OF THE ORGANISM IS Y, AND
3) IT IS DEFINITE (1) THAT THE AEROBICITY OF THE ORGANISM IS Z

Instead MYCIN contains a single rule of the following form:

RULE030

IF: THE IDENTITY OF THE ORGANISM IS KNOWN WITH CERTAINTY
THEN: IT IS DEFINITE (1) THAT THESE PARAMETERS - GRAM MORPH AIR - SHOULD BE TRANSFERRED FROM THE IDENTITY OF THE ORGANISM TO THIS ORGANISM

Thus if ORGANISM-1 is known to be a streptococcus, MYCIN can use RULE030 to access the knowledge table to look up the organism's gramstain, morphology, and aerobicity.

3.2.6-2 Specialized Functions

The efficient use of knowledge tables requires the existence of four specialized functions (the category <special-func> from § 3.2.1-2). As explained below, each function attempts to add members to a list named GRIDVAL and returns T if at least one element has been found to be placed in GRIDVAL.

Functions of the class <special-func>:

Let V=Val[C,P] be the set of all hypotheses regarding the value of the clinical parameter P for the context C.
Let CLST be a list of objects which may be characterized by clinical parameters.
Let PLST be a list of clinical parameters.
Then:

FUNCTION	Value of GRIDVAL
SAME2[C,CLST,PLST]	{X | X ∈ CLST & (for all P in PLST) SAME [C,P,Val[X,P]]}
NOTSAME2[C,CLST,PLST]	{X | X ∈ CLST & (for at least one P in PLST) NOTSAME[C,P,Val[X,P]]}
SAME3[C,P,CLST,P*]	{X | X ∈ CLST & SAME[C,P,Val[X,P*]]}
NOTSAME3[C,P,CLST,P*]	{X | X ∈ CLST & NOTSAME[C,P,Val[X,P*]]}
GRID[<object>, <attribute>]	{X | X is a value of the <attribute> of <object>}

GRID is merely a function for looking up information in the specialized knowledge table.

The use of these functions is best explained by example. Consider the following verbalization of a rule given us by one of our collaborating experts:

If you know the portal of entry of the current organism and also know the pathogenic bacteria normally associated with that site, you have evidence that the current organism is one of those pathogens so long as there is no disagreement on the basis of gramstain, morphology, or aerobicity.

This horrendous sounding rule is coded quite easily using SAME2[C,CLST,PLST], where C is the current organism, CLST is the list of pathogenic bacteria normally associated with the portal of

entry of C, and PLST is the set of properties (GRAM MORPH AIR). GRID is used to set up CLST. The LISP version of the rule is:

```
RULE084

PREMISE:  ($AND (GRID (VAL CNTXT PORTAL) PATH-FLORA)
                (SAME 2 CNTXT GRIDVAL
                      (QUOTE (GRAM MORPH AIR))))
ACTION:   (CONCLIST CNTXT IDENT GRIDVAL .8)
```

Note that GRID sets up the initial value of GRIDVAL for use by SAME2, which then redefines GRIDVAL for use in the ACTION clause. This rule is translated (in somewhat stilted English) as follows:

```
RULE084

IF:   1) THE LIST OF LIKELY PATHOGENS ASSOCIATED WITH
      THE PORTAL OF ENTRY OF THE ORGANISM IS KNOWN,
      AND
      2) THIS CURRENT ORGANISM AND THE MEMBERS YOU
      ARE CONSIDERING AGREE WITH RESPECT TO THE
      FOLLOWING PROPERTIES: GRAM MORPH AIR
THEN: THERE IS STRONGLY SUGGESTIVE EVIDENCE (.8) THAT
      EACH OF THEM IS THE IDENTITY OF THIS CURRENT
      ORGANISM
```

SAME2 and NOTSAME2 can also be used for comparing the values of the same clinical parameters for two or more different contexts in the context tree. For example:

```
SAME2[ORGANISM-1 (ORGANISM-2 ORGANISM-3) (GRAM MORPH)]
```

On the other hand, SAME3 and NOTSAME3 are useful for comparing different parameters of two or more contexts. Suppose you need a predicate that returns T if the site of a prior organism (ORGANISM-2) is the same as the portal of entry of the current organism (ORGANISM-1). This is accomplished by:

```
SAME3[ORGANISM-1 PORTAL (ORGANISM-2) SITE]
```

3.2.7 (*) TRANSLATION OF RULES INTO ENGLISH

Rules are translated into a subset of English using a set of recursive functions that piece together bits of text. I shall demonstrate the process using the PREMISE condition (GRID (VAL CNTXT PORTAL) PATH-FLORA) that is taken from RULE084 as discussed in § 3.2.6-2.

The reader will recall that every clinical parameter has a property named TRANS that is used for translation (§ 3.2.4-2). In addition, every function, simple list, or knowledge table that is used by MYCIN's rules also has a TRANS property. For our example the following TRANS properties are relevant:

GRID:	(THE (2) ASSOCIATED WITH (1) IS KNOWN)
VAL:	(((2 1)))
PORTAL:	(THE PORTAL OF ENTRY OF *)
PATH-FLORA:	(LIST OF LIKELY PATHOGENS)

The numbers in the translations of functions indicate where the translation of the corresponding argument should be inserted. Thus the translation of GRID's second argument is inserted for the "(2)" in GRID's TRANS property. The extra parentheses in the TRANS for VAL indicate that the translation of VAL's first argument should be substituted for the asterisk in the translation of VAL's second argument. Since PORTAL is a PROP-ORG, CNTXT translates as THE ORGANISM and the translation of (VAL CNTXT PORTAL) becomes:

THE PORTAL OF ENTRY OF THE ORGANISM

Substituting VAL's translation for the (1) in GRID's TRANS, and PATH-FLORA's translation for the (2), the final translation of the conditional clause becomes:

THE LIST OF LIKELY PATHOGENS ASSOCIATED WITH THE PORTAL OF ENTRY OF THE ORGANISM IS KNOWN

Similarly,

(GRID (VAL CNTXT CLASS) CLASSMEMBERS)

translates as:

THE LIST OF MEMBERS ASSOCIATED WITH THE CLASS OF THE ORGANISM IS KNOWN

All other portions of rules use essentially this same procedure for translation. An additional complexity arises, however, if it is necessary to negate the verbs in ACTION or ELSE clauses when the associated CF is negative. The translator program must therefore recognize verbs and know how to negate them when evidence in a PREMISE supports the negation of the hypothesis that is referenced in the ACTION of the rule.

3.3 Use of Rules to Give Advice

The discussion in § 3.2 was limited to the various data structures used to represent MYCIN's knowledge. The present section proceeds to an explanation of how MYCIN uses that knowledge in order to give advice.

The discussion begins with a summary of previous goal-oriented or rule-based problem-solving systems. I then describe MYCIN's control structure for selecting rules and deciding when to ask questions of the user. Subsequent sections explain the mechanisms for creation of the program's record of the consultation. They also describe a variety of nontrivial complexities that arose during implementation of the system's control structure.

3.3.1 PREVIOUS GOAL-ORIENTED PROBLEM-SOLVING SYSTEMS

Early AI research on machine reasoning concentrated on programs that could solve simple puzzles. From this work a number of problem-solving techniques were developed, many of which continue to pervade AI investigation. These have been summarized as follows [Nilsson, 1974]:

(1) heuristic search
(2) problem spaces and states

(3) operators for state transformations
(4) goal and subgoal states
(5) means-ends analysis
(6) reasoning backwards

I will not attempt to discuss all of these here, but will concentrate instead on the techniques used by the four "rule-based" systems that were selected for discussion in § 3.2.1-1 and on the various methodologies for goal-oriented problem-solving.

Although MYCIN shares its rule-based knowledge representation with several other AI programs, none of the systems described in § 3.2.1-1 uses its rules in the way that MYCIN does. Waterman's system, for example, makes decisions by comparing the current state vector with the "situation" portion of the SA rules [Waterman, 1970]. The rules are maintained in an ordered list and the matching-search begins with the first rule in the list. Searching stops as soon as a match is found; thus the first matched rule defines the program's "move" in the poker game. Subsequent rules in the list that might also match the current state vector are ignored. As a result, the order of rules in the rule-list is of crucial importance. In general, the most specific rules are placed early in the list so that they effectively filter out state vectors that are well-characterized and for which well-defined heuristics exist.

Although system knowledge is kept modular by the SA rule approach, the rules are implicitly interrelated by their ordering in the list. Furthermore, in HEURISTIC DENDRAL [Buchanan, 1969], the interrelationships may be explicit in that the action portion of one rule may include a pointer to one or more other rules. As a result, integration of new rules and modifications to old knowledge may be complicated. Waterman's program attempts to learn new heuristics for incorporation into the ordered list of rules, and META-DENDRAL [Buchanan, 1972], also tackles the problem of generalization (theory formation). Both programs must therefore select the appropriate location or mechanism for incorporating a new rule and, in some cases, must modify other rules so that the new SA heuristic will be invoked under appropriate circumstances.

Colby's system [Colby, 1969] interrelates its rules in a directed graph [Tesler, 1968]. In judging the credibility of a proposition P, the program looks for relevant beliefs in the graph structure. A

directly relevant belief is one that can be derived from P in a single step. These beliefs then serve as the "heads" of paths in the graph to be searched. Therefore, Colby's system clearly depends upon explicit interrelationships of both inferential rules and "facts" (see § 3.2.1-1). Furthermore, the program uses the rules primarily as a kind of pattern matching mechanism during the evaluation of the proposition in question. Despite its use of rules, the program is not really a problem-solving system and its similarity to MYCIN is therefore largely superficial.

Green's QA3, on the other hand, is a problem-solving system with a theoretical foundation firmly linked to the puzzle-solving programs that I mentioned above [Nilsson, 1974]. As explained in § 3.2.1-1, QA3's task is to use axioms and theorems (expressed in the first-order predicate calculus) to answer questions [Green, 1969]. Questions are themselves expressed as theorems (rules) and the program attempts to derive the theorem from its knowledge-base. The steps in the proof are remembered and then form the basis of the answer to the question. Thus the question (expressed as a theorem) is a "goal-statement" and the program must have mechanisms for selecting relevant pieces of knowledge that can be combined to accomplish the goal.

QA3's technique for combining knowledge is a modified form of the resolution principle [Robinson, 1965]. The principle explains how to derive a new logical statement, when possible, from a specified pair of clauses. However, a variety of additional strategies is needed for deciding which pieces of knowledge to attempt to resolve. Green's technique is to try to show that the negation of the question is inconsistent with the rest of the system's knowledge. Aided by heuristic search strategies including the set-of-support [Wos, 1965], unit preference [Wos, 1964] and subsumption [Robinson, 1965], QA3 works backwards from the negation of the question, attempting to derive a contradiction. Thus, this theorem-proving approach may be considered goal-oriented in that it works backwards from its goal rather than resolving knowledge clauses at random in hopes of eventually deriving the answer to the question under consideration.

Another intuitively pleasing technique that has found application within the realm of problem-solving [Fikes, 1971; Newell, 1961] is known as means–ends analysis. Often explained in terms of state transition, the technique is based upon the recognition of differences

between the current state of the system and the desired state (goal). As a result, useful intermediate states (subgoals) can be defined so that the problem may be reduced to a number of subproblems, each much easier than the total task. Plans for accomplishing each subgoal may then be combined to create a total strategy for achieving the goal.

It is not always natural to express knowledge in terms of operators for state transition, however. As early as 1957, a system was introduced to solve logical problems by working backwards from the goal without means–ends analysis [Newell, 1957]. More recent systems have also utilized the goal-oriented approach [Hewitt, 1969; Rulifson, 1972]. In fact, the consequent theorems of PLANNER [Hewitt, 1972] (implemented in MICRO-PLANNER; see also § 3.2.1-1), provide a control mechanism for knowledge use which seems strikingly similar to those that should ideally be used for medical decision making. I will attempt to justify this claim after a brief description of PLANNER's deductive mechanisms. The examples used here are taken from a recent discussion of AI languages [Bobrow, 1973].

PLANNER's data types include assertions, goals, and theorems. Consider, for example, a program that knew the following facts:

```
(PART ARM PERSON)
(PART HAND ARM)
(PART FINGER HAND)
```

where these stand for attribute-object-value triples such as those I discussed in § 3.2.3-1. Suppose the program were now asked to decide whether a finger is part of a person, i.e.:

```
(GOAL (PART FINGER PERSON))
```

The PLANNER "GOAL" formalism first looks to see if the fact appears in the program's knowledge-base. Since it does not, it looks instead for a "consequent theorem" with a pattern that matches the GOAL statement (PART FINGER PERSON). Variable positions in patterns are characters preceded by '$?'. Thus the following consequent theorem matches the GOAL:

```
(CONSEQUENT
        (PART $?X $?Z)          ←(pattern)
        (GOAL (PART $?X $?Y))
        (GOAL (PART $?Y $?Z)))
```

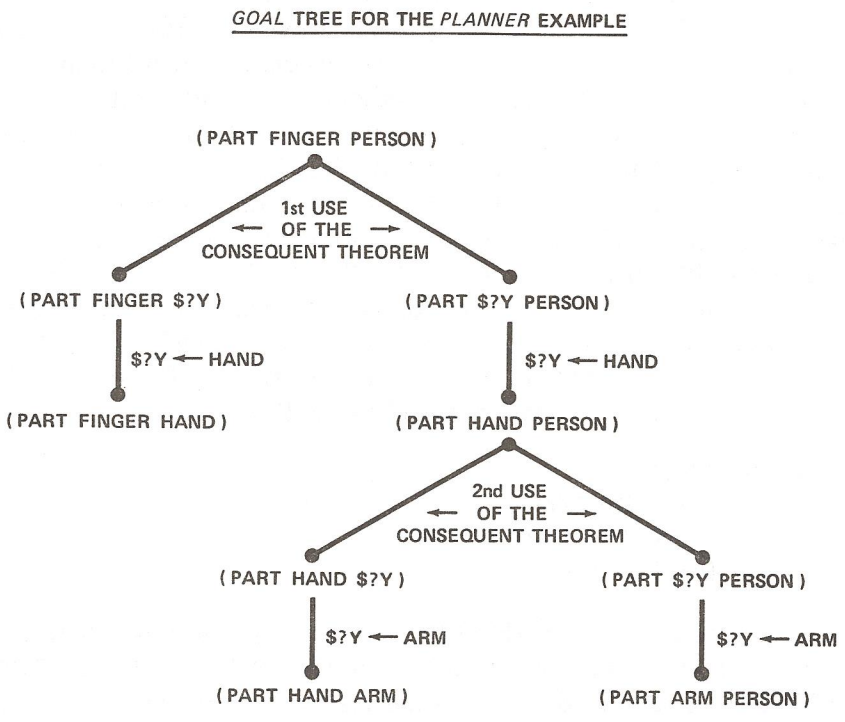


Figure 3-6: This goal tree corresponds to the example of PLANNER reasoning strategies described in the text. The terms $?Y and $?Z correspond to variable positions in the PLANNER patterns.

When instantiated for the current GOAL, the theorem becomes:

```
(CONSEQUENT
        (PART FINGER PERSON)
        (GOAL (PART FINGER $?Y))
        (GOAL (PART $?Y PERSON)))
```

or, in words, to show that a finger is part of a person, find something ($?Y) of which a finger is a part and which itself is a part of a person. Thus the program has two new instantiated GOAL statements:

```
(GOAL (PART FINGER $?Y))
(GOAL (PART $?Y PERSON))
```

The first GOAL statement immediately finds from its knowledge base that (PART FINGER $?Y) holds for $?Y = HAND. Thus the second GOAL becomes (GOAL (PART HAND PERSON)); this can, in turn, be derived by recursive use of the consequent theorem given above. Figure 3-6 diagrams the reasoning network that develops below the initial GOAL. Note that the terminal nodes in this little tree correspond to facts already in the data base. As is later shown, MYCIN's decision process may also be diagrammed as a reasoning network with a goal at the top and known data as terminal nodes.

Another PLANNER construct is the "antecedent theorem." Whenever anything is asserted in a PLANNER program, i.e., added to the data base, the system compares the new knowledge with the pattern portion of all antecedent theorems in the system. Continuing the example from above, consider the following theorem:

```
(ANTECEDENT
        (PART $?X $?Y)              ←(pattern)
        (GOAL (PART $?Y $?Z))
        (ASSERT (PART $?X $?Z)))
```

Suppose the assertion (PART FINGER HAND) were now added to a program that already knew (PART HAND ARM) and (PART ARM PERSON). The new assertion would match the pattern in the theorem above (note I have not yet mentioned any GOALs) and would therefore invoke the following instantiation:

```
(ANTECEDENT
    (PART FINGER HAND)
    (GOAL (PART HAND $?Z))
    (ASSERT (PART FINGER $?Z)))
```

which says, in words, that since a finger is part of a hand, if you can find something ($?Z) which a hand is part of then you can assert that a finger is part of it too. (GOAL (PART HAND $?Z)) is in this case easily proven from the data base by setting $?Z to ARM. Thus the antecedent theorem succeeds and asserts (PART FINGER ARM). However, this new assertion also matches the pattern portion of the antecedent theorem, so the theorem is once again invoked. This time the observation (PART ARM PERSON) leads to the conclusion (PART FINGER PERSON).

A potential problem with antecedent theorems, as should be clear from this example, is that they have a capability to clutter up the system's knowledge-base with facts (assertions) that will never be used in achieving goals. When used judiciously they are powerful mechanisms for simplifying future goals that are likely to need the generated assertions, but the consequent theorems suggest a sense of purpose which is highly appealing for problem-solving applications.

The distinction between consequent and antecedent theorems provides a useful basis for considering some of the different approaches to the medical diagnosis problem. Antecedent theorems may in one sense be compared with a comprehensive process for medical data collection. Clinical screening exams, of course, have their place (§ 1.2.2-3), but medical education tends to stress the rational selection of tests based upon indications in the patient. The alternate approach is to order every test imaginable (including a lengthy history and physical exam) and then to sift through the data in hopes of recognizing unusual patterns or clusters of symptoms that may lead to a diagnosis. The second alternative is not only expensive and time-consuming, but it also requires remarkably little analytical skill on the part of the clinician. The approach does occur, however, particularly among medical students before their clinical skills are well-developed.

The selection of tests on the basis of specific indications, on the other hand, indicates an organized approach to problem-solving that parallels that found in consequent theorems. The good clinician tends to work backwards from his goal (i.e., to diagnose and treat his patient appropriately), making hypotheses and selecting tests in accordance with his desire to minimize unnecessary time-delays or monetary expenditures. This comparison to PLANNER-type consequent theorems may at first seem rather vague, but I shall show in subsequent sections that MYCIN indeed does reason backwards, avoiding the "shotgun approach" of a diagnostic system based solely upon mechanisms analogous to antecedent theorems.

3.3.2 MYCIN'S CONTROL STRUCTURE

MYCIN's rules are directly analogous to the PLANNER consequent theorems discussed in § 3.3.1. They permit a reasoning chain (see Figure 3-6) to grow dynamically on the basis of the user's

answers to questions regarding the patient. In this subsection, I describe that reasoning network, explaining how it grows and how MYCIN manages to ask questions only when there is a reason for doing so.

3.3.2-1 Consequent Rules and Recursion

As discussed in § 1.4.1, MYCIN's task involves a four-stage decision problem:

(1) Decide which organisms, if any, are causing significant disease;
(2) Determine the likely identity of the significant organisms;
(3) Decide which drugs are potentially useful;
(4) Select the best drug or drugs.

Steps 1 and 2 are closely interrelated since determination of an organism's significance may well depend upon its presumed identity. Furthermore, MYCIN must consider the possibility that the patient has an infection with an organism not specifically mentioned by the user (e.g., an occult abscess suggested by historical information or subtle physical findings). Finally, if MYCIN decides that there is no significant infection requiring antimicrobial therapy, it should skip steps 3 and 4, advising the user that no treatment is thought to be necessary. MYCIN's task area therefore can be defined by the following rule:

RULE092

IF: 1) THERE IS AN ORGANISM WHICH REQUIRES THERAPY, AND
2) CONSIDERATION HAS BEEN GIVEN TO THE POSSIBLE EXISTENCE OF ADDITIONAL ORGANISMS REQUIRING THERAPY, EVEN THOUGH THEY HAVE NOT ACTUALLY BEEN RECOVERED FROM ANY CURRENT CULTURES
THEN: DO THE FOLLOWING:
1) COMPILE THE LIST OF POSSIBLE THERAPIES WHICH, BASED UPON SENSITIVITY DATA, MAY BE EFFECTIVE AGAINST THE ORGANISMS REQUIRING TREATMENT, AND

2) DETERMINE THE BEST THERAPY
RECOMMENDATIONS FROM THE COMPILED LIST
OTHERWISE: INDICATE THAT THE PATIENT DOES NOT REQUIRE THERAPY

This rule is one of MYCIN's PATRULES (i.e., its context is the patient; see § 3.2.2-2) and is known as the "goal rule" for the system. A consultation session with MYCIN results from a simple two-step procedure (Subprogram 1 shown in Figure 1-1):

(1) Create the patient context as the top node in the context tree (see § 3.4 for an explanation of how nodes are added to the tree)
(2) Attempt to apply the goal-rule to the newly created patient context

After the second step, the consultation is over and Subprogram 1 relinquishes control to the Explanation System (Subprogram 2 shown in Figure 1-1). My purpose here, then, is to explain how the simple attempt to apply the goal rule to the patient causes a lengthy consultation with an individualized reasoning chain.

When MYCIN first tries to evaluate the PREMISE of the goal rule, the first condition requires that it know whether there is an organism that requires therapy. MYCIN then reasons backwards in a manner that may be informally paraphrased as follows:

How do I decide whether there is an organism requiring therapy? Well, RULE090 tells me that organisms associated with significant disease require therapy. But I don't even have any organisms in the context tree yet, so I'd better ask first if there are any organisms and if there are I'll try to apply RULE090 to each of them. However, the PREMISE of RULE090 requires that I know whether the organism is significant. I have a bunch or rules for making this decision (RULE038 RULE042 RULE044 RULE108 RULE122). For example, RULE038 tells me that if the organism came from a sterile site it is probably significant. Unfortunately I don't have any rules for inferring the site of a culture, however, so I guess I'll have to ask the user for this information when I need it . . .

This goal-oriented approach to rule invocation and question selection is automated via two interrelated procedures, a MONITOR that

THE *MONITOR* FOR RULES

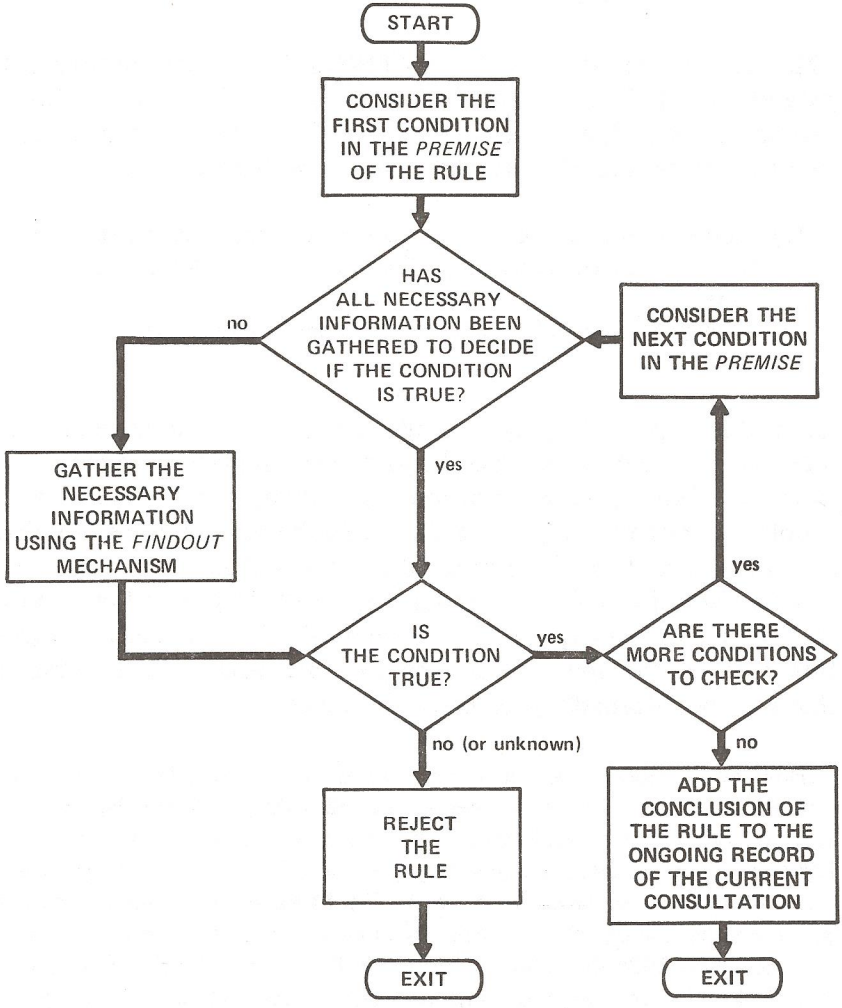


Figure 3-7: Flow chart describing the rule MONITOR that analyzes a rule and decides whether it applies in the clinical situation under consideration. Each condition in the PREMISE of the rule references some clinical parameter, and all such conditions must be true for the rule to be accepted. [Reproduced from *Computers and Biomedical Research* [Shortliffe, 1975b] with permission of the publishers.]

analyzes rules and a FINDOUT mechanism that searches for data needed by the MONITOR.

The MONITOR analyzes the PREMISE of a rule, condition by condition, as shown in Figure 3-7. (As discussed in § 3.2.5, the MONITOR uses the $AND function to oversee the PREMISE evaluation.) When the value of the clinical parameter referenced in a condition is not yet known to MYCIN, the FINDOUT mechanism is

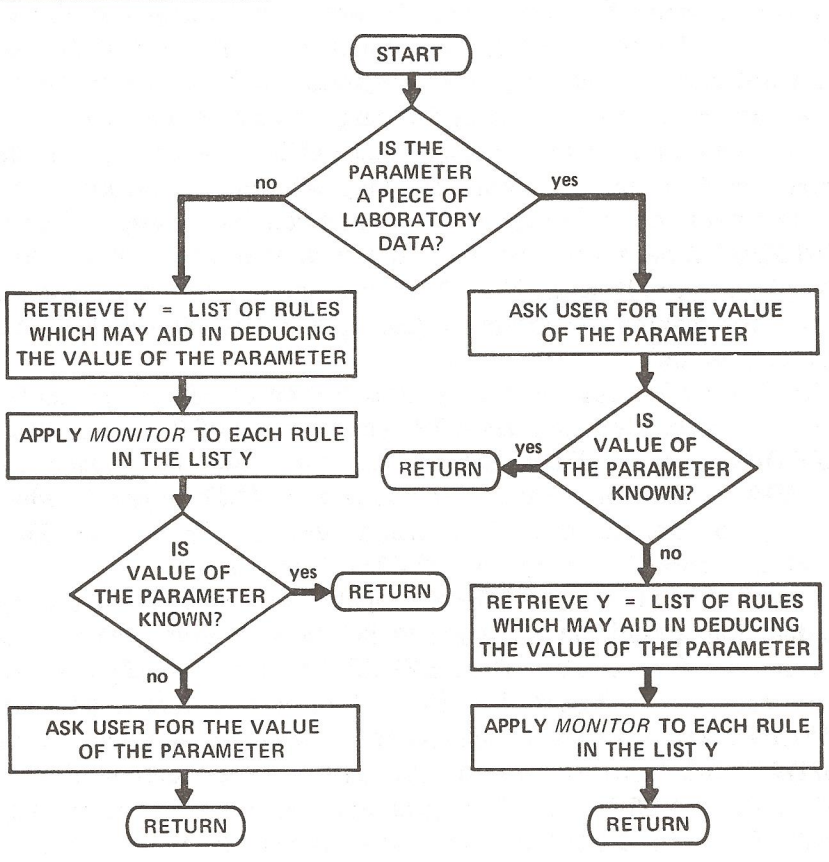


Figure 3-8: Flow chart describing the strategy for determining which questions to ask the physician. The derivation of values of parameters may require recursive calls to the MONITOR, thus dynamically creating a reasoning chain specific to the patient under consideration. [Reproduced from *Computers and Biomedical Research* [Shortliffe, 1975b] with permission of the publishers.]

invoked in an attempt to obtain the missing information. FINDOUT then either derives the necessary information (from other rules) or asks the user for the data.

FINDOUT has a dual strategy depending upon the kind of information required by the MONITOR. This distinction is demonstrated in Figure 3-8. In general, a piece of data is immediately requested from the user (an ASK1 question) if it is considered in some sense "primitive," as are, for example, most laboratory data. Thus, if the physician knows the identity of an organism (e.g., from a lab report), we would prefer that the system request that information directly rather than try to deduce it via decision rules. However, if the user does not know the identity of the organism, MYCIN uses its knowledge base in an effort to deduce the range of likely organisms.

"Nonlaboratory data" are those kinds of information that require inference even by the clinician: e.g., whether an organism is a contaminant or a previously administered drug was effective. FINDOUT always attempts to deduce such information first, asking the physician only when MYCIN's knowledge-base of rules is inadequate for making the inference from the information at hand (an ASK2 question).

In § 3.2.3-2 I described the representation of clinical parameters and their associated properties. The need for two of these properties, LABDATA and UPDATED-BY, should now be clear. The LABDATA flag for a parameter allows FINDOUT to decide which branch to take through its decision process (Figure 3-8). Thus, IDENT is marked as being LABDATA in Figure 3-2.

Recall that the UPDATED-BY property is a list of all rules in the system that permit an inference to be made regarding the value of the indicated parameter. Thus, UPDATED-BY is precisely the list I have called Y in Figure 3-8. Every time a new rule is added to MYCIN's knowledge-base, the name of the rule is added to the UPDATED-BY property of the clinical parameter referenced in its ACTION or ELSE clause. Thus, the new rule immediately becomes available to FINDOUT at times when it may be useful. It is not necessary explicitly to specify its interrelationships with other rules in the system.

Note that FINDOUT is accessed from the MONITOR, but the MONITOR may also be accessed from FINDOUT. This recursion allows self-propagation of a reasoning network appropriate for the

patient under consideration and selects only the necessary questions and rules. The first rule passed to the MONITOR is always the goal rule. Since the first condition in the PREMISE of this rule references a clinical parameter of the patient named TREATFOR, and since the value of TREATFOR is, of course, unknown before any data have been gathered, the MONITOR asks FINDOUT to trace the value of TREATFOR. This clinical parameter is not a LABDATA so FINDOUT takes the left-hand pathway in Figure 3-8 and sets Y to the UPDATED-BY property of TREATFOR, the two-element list (RULE090 RULE149). The MONITOR is then called again with RULE090 as the rule for consideration, and FINDOUT is utilized to trace the values of clinical parameters referenced in the PREMISE of RULE090. Note that this process parallels the verbal description of MYCIN's reasoning that was given above. (The reference to tree propagation, however, will not be explained until § 3.4.)

It is important to recognize that FINDOUT does not check to see whether the PREMISE condition is true. Instead the FINDOUT mechanism traces the clinical parameter exhaustively and returns its value to the MONITOR where the conditional expression may then be evaluated. (The process is slightly different for "multivalued" parameters; see § 3.3.2-2.) Hence FINDOUT is called at most one time for a clinical parameter (in a given context, see § 3.4). When FINDOUT returns a value to the MONITOR it marks the clinical parameter as having been traced. Thus, when the MONITOR reaches the question "HAS ALL NECESSARY INFORMATION BEEN GATHERED TO DECIDE IF THE CONDITION IS TRUE?" (Figure 3-7), the parameter is immediately passed to FINDOUT unless it has been previously marked as traced.

Figure 3-9 is a portion of MYCIN's initial reasoning chain. A comparison with Figure 3-6 will reemphasize the similarities between MYCIN's control structure and the goal-oriented consequent theorems used by PLANNER. In Figure 3-9 the clinical parameters being traced are underlined. Thus REGIMEN is the top goal of the system (i.e., it is the clinical parameter in the ACTION clause of the goal rule). Below each parameter are the rules (from the UPDATED-BY property) which may be used for inferring the parameter's value. Clinical parameters referenced in the PREMISE of these rules are then listed at the next level in the reasoning network. Rules with multiple PREMISE conditions have their links numbered in accor-

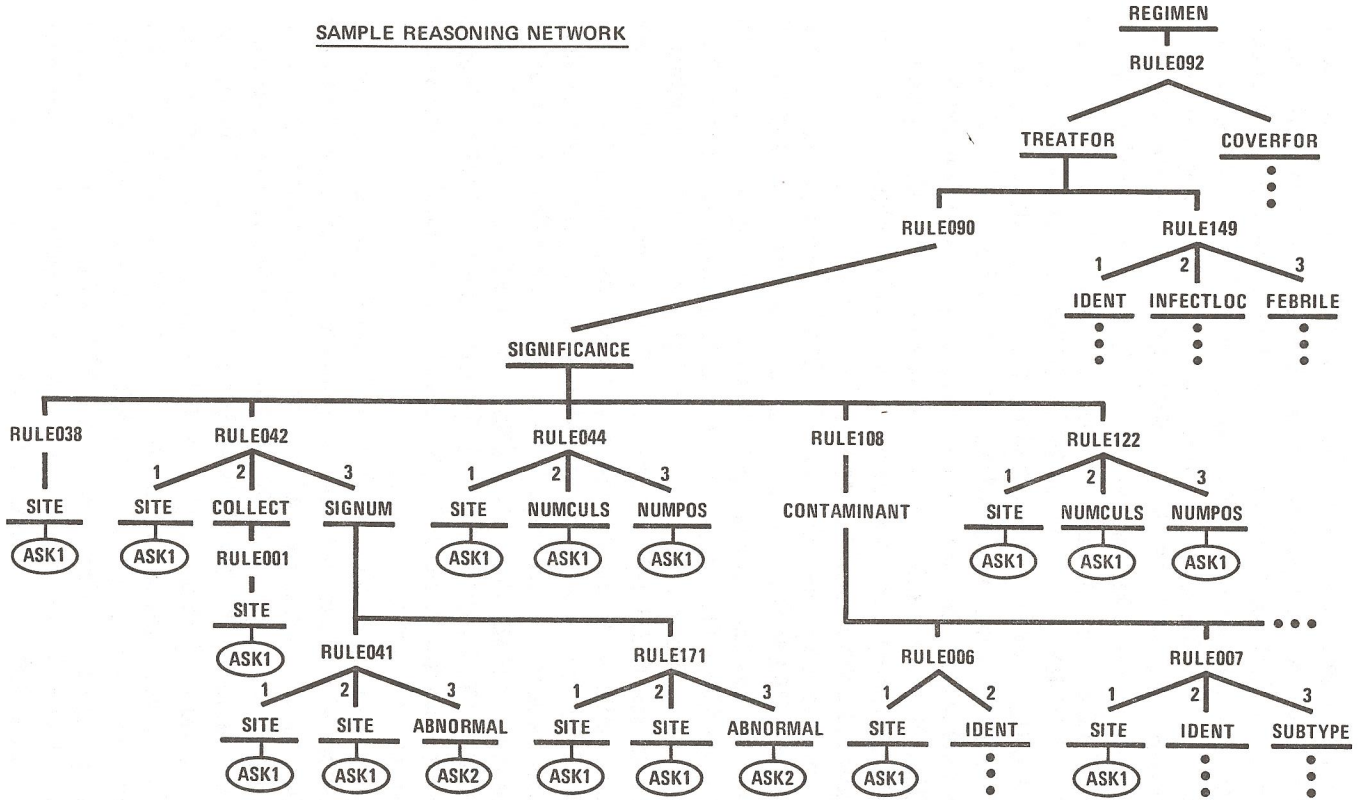


Figure 3-9: An example of the kind of reasoning network generated by the MONITOR and FINDOUT mechanisms. Names of clinical parameters are underlined. When rules have multiple conditions in their PREMISE, a number has been included to specify the position of the associated clinical parameter within the PREMISE conditions.

dance with the order in which the parameters are traced (by FINDOUT). ASK1 indicates that a parameter is LABDATA so its value is automatically asked of the user when it is needed. ASK2 refers to parameters that are not LABDATA but for which no inference rules currently exist, e.g., whether the dose of a drug is adequate. One of the goals in the future development of MYCIN's knowledge base is to acquire enough rules allowing the values of non-LABDATA parameters to be inferred so that ASK2 questions need no longer occur.

Note that the reasoning network in Figure 3-9 is drawn to reflect maximum size. In reality many portions of such a network need not be considered. For example, RULE042 (one of the UPDATED-BY rules under SIGNIFICANCE) is rejected if the SITE condition is found to be false by the MONITOR. When that happens, neither COLLECT nor SIGNUM need to be traced by FINDOUT and those portions of the reasoning network are not created. Thus, the order of conditions within a PREMISE is highly important. In general, conditions referencing the parameters that are most common (i.e., which appear in the PREMISE of the most rules) are put first in the PREMISE of new rules to act as an effective screening mechanism.

A final comment is necessary regarding the box labelled "REJECT THE RULE" in Figure 3-7. This step in the MONITOR actually must check to see if the rule has an ELSE clause. If so, and if the PREMISE is known to be false, the conclusion indicated by the ELSE clause is drawn. If there is no ELSE clause, or if the truth status of the PREMISE is uncertain (e.g., the user has entered UNKNOWN when asked the value of one of the relevant parameters, see § 3.3.2-2), the rule is simply ignored.

3.3.2-2 Asking Questions of User

As was emphasized in Chapter 2, the conventions for communication between a program and the physician are a primary factor determining the system's acceptability. We have therefore designed a number of features intended to simplify the interactive process that occurs when FINDOUT reaches one of the boxes entitled "ASK THE USER FOR THE VALUE OF THE PARAMETER" (Figure 3-8).

When MYCIN requests the value of a "single-valued" or "yes–no" parameter, it uses the PROMPT property as described in § 3.2.3-2.

The user's response is then compared with the EXPECT property of the parameter. If his answer is one of the expected responses, the program simply continues through the reasoning network. Otherwise, MYCIN checks the system dictionary to see if the user's response is a synonym for one of the recognized answers. If this attempt also fails, MYCIN uses INTERLISP spelling-correction routines [Teitelman, 1974] to see if a simple spelling or typographical error will account for the unrecognized response. If so, the program makes the correction, prints its assumption, and proceeds as though the user had made no error. If none of these mechanisms succeeds, MYCIN tells the user that his response is not recognized, displays a list of sample responses, and asks the question again. Examples of these features are included in the sample consultation session at the end of Chapter 1.

"Multivalued" parameters are handled somewhat differently. FINDOUT recursively traces such parameters in the normal fashion, but when forced to ask a question of the user it customizes its question to the condition being evaluated in the MONITOR. Suppose, for example, the MONITOR were evaluating the condition (SAME CNTXT INFECT MENINGITIS), i.e., "Meningitis is an infectious disease diagnosis for the patient." If FINDOUT were to ask the question using the regular PROMPT strategy, it would request:

"What is the infectious disease diagnosis for PATIENT-1?"

The problem is that the patient may have several diagnoses, each of which can be expressed in a variety of ways. If the physician were to respond:

"A meningeal inflammation that is probably of infectious origin"

MYCIN would be forced to try to recognize that this answer implies meningitis. Our solution has been to customize questions for "multivalued" parameters to reflect the value being checked in the current PREMISE condition. The PROMPT1 property is used, and questions always expect a yes–or–no response:

"Is there evidence that the patient has a meningitis?"

The advantages of this approach are the resulting ability to avoid natural language processing during the consultation itself, and the posing of questions that are specific to the patient under consideration.

In addition to the automatic spelling-correction capability described above, the user is given a number of options that may be utilized whenever MYCIN asks him a question:

UNKNOWN	-	(may be abbreviated U or UNK) used to indicate that the physician does not know the answer to the question, usually because the data are unavailable.
?	-	used to request a list of sample recognized responses.
??	-	used to request a list of all recognized responses.
RULE	-	used to request that MYCIN display the translation of the current decision rule. FINDOUT simply translates the rule being considered by the MONITOR. This feature provides a simple capability for explaining why the program is asking the question. However, it cannot explain motivation beyond the current decision rule.
QA	-	used to digress temporarily in order to use the Explanation System (Subprogram 2). The features of this system are explained in Chapter 5.
WHY	-	used to request a detailed explanation of the question being asked. This feature is much more conversational that the RULE option above and permits investigation of the current state of the entire reasoning chain. This explanation capability is described elsewhere [Shortliffe, 1975b; Davis, 1976].
CHANGE XXX	-	used to change the answer to a previous question. Whenever MYCIN asks a question it prints a number in front of the prompt. Thus CHANGE 4 means "Go back and let me reanswer question #4". The complexities involved in this process are discussed in § 3.6.1.
STOP	-	halts the program without completing the consultation.
HELP	-	prints this list.

3.3.3 CREATION OF DYNAMIC DATA BASE

Figure 1-1 showed that the Consultation System maintains an ongoing record of the consultation. These dynamic data include

information entered by the user, inferences drawn using decision rules, and record-keeping data structures that facilitate question answering by the Explanation System (see Chapter 5).

3.3.3-1 Data Acquired from User

Except for questions related to propagation of the context tree, all queries from MYCIN to the physician request the values of specific clinical parameters for specific nodes in the context tree. The FINDOUT mechanism screens the user's response, as described in § 3.3.2-2, stores it in MYCIN's dynamic data base, and returns the value to the MONITOR for evaluation of the conditional statement that generated the question in the first place (§ 3.3.2-1). The physician's response is stored, of course, so that future rules containing conditions referencing the same clinical parameter will not force the question to be asked a second time.

As we noted in § 3.2.4, however, the values of clinical parameters are always stored along with their associated certainty factors. A physician's response must therefore have a CF associated with it. MYCIN's convention is to assume CF=1 for the response unless the physician explicitly states otherwise. Thus the following exchange:

```
7) Staining characteristics of ORGANISM-1 (gram):
**GRAMNEG
```

results in:

```
Val[ORGANISM-1,GRAM] = ((GRAMNEG 1.0))
```

If, on the other hand, the user thinks he knows the answer to a question but wants to indicate his uncertainty, he may enter a certainty factor in parentheses after his response. MYCIN expects the number to be an integer between −10 and +10; the program divides the number by 10 to obtain a CF. Using integers simplifies the user's response and also discourages comparisons between the number and a probability measure. Thus the following exchange:

```
8) Enter the identity (genus) of ORGANISM-1:
**ENTEROCOCCUS (8)
```

results in

```
Val[ORGANISM-1,IDENT] = ((STREPTOCOCCUS-GROUP-D .8))
```

This example also shows how the dictionary is used to put synonyms into standardized form for the patient's data base (i.e., enterococcus is effectively another name for a group-D streptococcus).

A variant of this last example is the user's option to enter multiple responses to a question so long as each is modified by a CF. For example:

```
13) Did ORGANISM-2 grow in clumps, chains, or pairs?
**CLUMPS (6) CHAINS (3) PAIRS (-8)
```

results in

```
Val[ORGANISM-2,CONFORM] = ((CLUMPS .6) (CHAINS .3) (PAIRS -.8))
```

The CF's associated with the parameter values are then used for evaluation of PREMISE conditions as described in § 3.2.5. Note that the user's freedom to modify his answers increases the flexibility of MYCIN's reasoning. Without the CF option, the user might well have responded UNKNOWN to question 13 above. The demonstrated answer, although uncertain, gives MYCIN much more information than would have been provided by an UNKNOWN.

3.3.3-2 Data Inferred by System

This subsection explains the <conclusion> item from the BNF rule description (§ 3.2.1-2), i.e., the functions that are used in ACTION or ELSE clauses when a PREMISE has shown that an indicated conclusion may be drawn. There are only three such functions, two of which (CONCLIST and TRANSLIST) reference knowledge tables (§ 3.2.6) but are otherwise dependent upon the third, a function called CONCLUDE. CONCLUDE takes five arguments:

CNTXT - the node in the context tree about which the conclusion is being made

PARAM - the clinical parameter whose value is being added to the dynamic data base

VALUE - the inferred value of the clinical parameter
TALLY - the certainty tally for the PREMISE of the rule (see § 3.2.4)
CF - the certainty factor for the rule as judged by the expert from whom the rule was obtained

The translation of CONCLUDE depends upon the size of CF:

CF $\geqslant$ .8 - "There is strongly suggestive evidence that . . ."
.4 $\leqslant$ CF $<$ .8 - "There is suggestive evidence that . . ."
CF $<$ .4 - "There is weakly suggestive evidence that . . ."
Computed CF - "There is evidence that . . ."

Thus the following conclusion:

(CONCLUDE CNTXT IDENT STREPTOCOCCUS TALLY .7)

translates as:

THERE IS SUGGESTIVE EVIDENCE (.7) THAT THE IDENTITY OF THE ORGANISM IS STREPTOCOCCUS

If, for example, the rule with this ACTION clause were successfully applied to ORGANISM-1, an organism for which no previous inferences had been made regarding identity, the result would be:

Val[ORGANISM-1,IDENT] = ((STREPTOCOCCUS X))

where X is the product of .7 and TALLY (see Combining Function 4, § 4.6). Thus the strength of the conclusion reflects both the CF for the rule *and* the extent to which the PREMISE of the rule is believed to be true for ORGANISM-1.

Suppose a second rule were now found which contained a PREMISE true for ORGANISM-1 and which added additional evidence to the assertion that the organism is a streptococcus. This new evidence somehow has to be combined with the CF (=X) that is already stored for the hypothesis that ORGANISM-1 is a streptococcus. If Y is the CF calculated for the second rule (i.e., the product of the TALLY for that rule and the CF assigned to the rule by the expert), the CF for the hypothesis is updated to Z so that:

Val[ORGANISM-1,IDENT] = ((STREPTOCOCCUS Z))

where Combining Function 1 gives $Z = X + Y(1-X)$. This function is justified and discussed in detail in § 4.6.

Similarly, additional rules leading to alternate hypotheses regarding the identity of ORGANISM-1 may be successfully invoked. The new hypotheses, along with their associated CF's, are simply appended to the list of hypotheses in Val[ORGANISM-1,IDENT]. Note, of course, that the CF's of some hypotheses may be negative, indicating there is evidence suggesting that the hypothesis is not true. When there is both positive and negative evidence for a hypothesis, Combining Function 1 must be used in a modified form. See Chapter 4 for these details, especially § 4.7 where MYCIN's use of the CF model is discussed with an example.

A final point to note is that values of parameters are stored identically regardless of whether the information has been inferred or acquired from the user (§ 3.3.3-1). The source of a piece of information is maintained in a separate record (§ 3.3.3-3). It is therefore easy to incorporate new rules that infer values of parameters for which ASK2 questions to the user once were necessary.

3.3.3-3 Creating an Ongoing Consultation Record

In addition to information provided or inferred regarding nodes in the context tree, MYCIN's dynamic data base contains a record of the consultation session. This record provides the basis for answering questions about the consultation (Chapter 5).

There are two general types of records kept. One is information about how values of clinical parameters were obtained. If the value was inferred using rules, a record of those inferences is stored with the rules themselves. Thus whenever an ACTION or ELSE clause is executed, MYCIN keeps a record of the details.

The second record provides a mechanism for explaining why questions were asked. MYCIN maintains a list of questions, their identifying number, the clinical parameter and context involved, plus the rule that led to generation of the question. The program then uses this list in responding to the EQ option (see Chapter 5) during interactive sessions between the physician and Subprogram 2.

3.3.4 (*) SELF-REFERENCING RULES

As new rules were acquired from the collaborating experts, it became apparent that MYCIN would need a small number of rules that departed from the strict modularity to which we had otherwise been able to adhere. For example, one expert indicated that he would tend to ask about the typical pseudomonas-type skin lesions only if he already had reason to believe that the organism was a pseudomonas. If the lesions were then said to be evident, however, his belief that the organism was a pseudomonas would be increased even more. A rule reflecting this fact must somehow imply an orderedness of rule invocation, i.e., "Don't try this rule until you have already traced the identity of the organism by using other rules in the system". Our solution has been to reference the clinical parameter early in the PREMISE of the rule as well as in the ACTION. For example:

RULE040

IF: 1) THE SITE OF THE CULTURE IS BLOOD, AND
2) THE IDENTITY OF THE ORGANISM MAY BE PSEUDOMONAS, AND
3) THE PATIENT HAS ECTHYMA GANGRENOSUM SKIN LESIONS

THEN: THERE IS STRONGLY SUGGESTIVE EVIDENCE (.8) THAT THE IDENTITY OF THE ORGANISM IS PSEUDOMONAS

Note that RULE040 is thus a member of both the LOOKAHEAD property and the UPDATED-BY property for the clinical parameter IDENT. Rules with the same parameter in both PREMISE and ACTION are termed "self-referencing" rules. The ordered invocation of such rules is accomplished by a generalized procedure described below.

As discussed in § 3.3.2-1, a rule such as RULE040 is originally invoked because MYCIN is trying to infer the identity of an organism, i.e., FINDOUT is asked to trace the parameter IDENT and recursively sends the UPDATED-BY list for that parameter to the MONITOR. When the MONITOR reaches RULE040, however, the second PREMISE condition references the same clinical parameter

currently being traced by FINDOUT. If the MONITOR merely passed IDENT to FINDOUT again (as called for by the simplified flow chart in Figure 3-7), FINDOUT would begin tracing IDENT for a second time, RULE040 would be passed to the MONITOR yet again, and an infinite loop would occur.

The solution to this problem is to let FINDOUT screen the list I call Y in Figure 3-8, i.e., the UPDATED-BY property for the parameter it is about to trace. Y is partitioned by FINDOUT into regular rules and self-referencing rules (where the latter category is defined as those rules that also occur on the LOOKAHEAD list for the clinical parameter). FINDOUT passes the first group of rules to the MONITOR in the normal fashion. After all these rules have been tried, FINDOUT marks the parameter as having been traced and then passes the self-referencing rules to the MONITOR. In this way, when the MONITOR considers the second condition in the PREMISE of RULE040, the conditional is evaluated without a call to FINDOUT because the parameter has already been marked as traced. Thus, the truth of the PREMISE of a self-referencing rule is determined on the basis of the set of non-self-referencing rules that were first evaluated. If one of the regular rules permitted MYCIN to conclude that an organism might be a pseudomonas, RULE040 might well succeed when passed to the MONITOR. Clearly this mechanism for handling self-referencing rules satisfies the intention of an expert when he gives us decision criteria in self-referencing form.

It should be noted that this approach minimizes the potential for self-referencing rules to destroy certainty factor commutativity. By holding these rules to the last we insure that the certainty tally for their PREMISE (see TALLY, § 3.2.5) is the same regardless of the order in which the non-self-referencing rules were executed. If there is more than one self-referencing rule that is successfully executed for a given context and parameter, however, the order of their invocation may affect the final CF. The approach we have currently implemented thus seeks merely to minimize the potential inconsistent effects of self-referencing rules.

3.3.5 (*) PREVENTING REASONING LOOPS

Self-referencing rules are actually a special case of a more general problem. Reasoning loops involving multiple rules cannot be handled

by the mechanism described in § 3.3.4. The difference is that self-referencing rules are intentional parts of MYCIN's knowledge base whereas reasoning loops are artifacts that must somehow be avoided.

For the following discussion, I introduce the following notation:

[q] X : :> Y

means that decision rule "q" uses clinical parameter X to reach a conclusion regarding the value of clinical parameter Y. Thus, a self-referencing rule may be represented by:

[a] E : :> E

where E is the clinical parameter that is referenced in both the PREMISE and the ACTION of the rule. Consider now the following set of rules:

[1] A : :> B
[2] B : :> C
[3] C : :> D
[4] D : :> A

Statement [1], for example, says that under certain unspecified conditions, the value of A can be used to infer the value of B. Now suppose that the MONITOR asks FINDOUT to trace the clinical parameter D. Then MYCIN's recursive mechanism would create the following reasoning chain:

[4] [1] [2] [3]
. . . D : :> A : :> B : :> C : :> D

The difference between this looped reasoning chain and a self-referencing rule is that rule [4] was provided as a mechanism for deducing the value of A, not for reinforcing the system's belief in the value of D. In cases where the value of A is of primary interest, the use of rule [4] would be appropriate. MYCIN solves this problem by keeping track of all parameters currently being traced by the FINDOUT mechanism. The MONITOR then simply ignores a rule if one of the parameters checked in its PREMISE is already being

traced. The result, with the value of D as the goal, is a three-membered reasoning chain in the case above:

[1] [2] [3]
A : :> B : :> C : :> D

Rule [4] is rejected because parameter D is already being traced elsewhere in the current reasoning chain. If the value of A were the main goal, however, the chain would be:

[2] [3] [4]
B : :> C : :> D : :> A

Note that this simple mechanism allows us to have potential reasoning loops in the knowledge-base but to select only the relevant nonlooping portions for consideration of a given patient.

A similar problem can occur when a rule permits two conclusions to be made, each about a different clinical parameter. MYCIN prevents loops in such circumstances by refusing to permit the same rule to occur twice in the current reasoning chain.

3.4 Propagation of Context Tree

The mechanism by which the context tree is customized for a given patient has not yet been discussed. As described in § 3.3.2-1, the consultation program begins simply by creating the patient context and then attempting to execute the goal rule. All additional nodes in the context tree are thus added automatically during the unwinding of MYCIN's reasoning regarding the PREMISE of the goal rule. This section first explains the data structures used for creating new nodes; then mechanisms for deciding when new nodes should be added are discussed.

3.4.1 DATA STRUCTURES USED FOR SPROUTING BRANCHES

Section 3.2.2-1 was devoted to an explanation of the context tree. At that time, I described the different kinds of contexts and explained that each node in the tree is an instantiation of the appropri-

ate context-type. Each context-type is characterized by the following properties:

PROMPT1	-	a sentence used to ask the user whether the first node of this type should be added to the context tree; expects a "yes–no" answer.
PROMPT2	-	a sentence used to ask the user whether subsequent nodes of this type should be added to the context tree.
PROMPT3	-	replaces PROMPT1 when it is used; this is a message to be printed out if MYCIN assumes that there is at least one node of this type in the tree.
PROPTYPE	-	indicates the category of clinical parameters (see § 3.2.3-2) that may be used to characterize a context of this type.
SUBJECT	-	indicates the categories of rules that may be applied to a context of this type.
SYN	-	indicates a conversational synonym for referring to a context of this type; MYCIN uses SYN when filling in the asterisk of PROMPT properties for clinical parameters.
TRANS	-	used for English translations of rules referencing this type of context.
TYPE	-	indicates what kind of internal name to give a context of this type.
MAINPROPS	-	lists the clinical parameters, if any, that are to be automatically traced (by FINDOUT) whenever a context of this type is created.
ASSOCWITH	-	gives the context-type of nodes in the tree immediately above contexts of this type.

Two sample context-types are shown in Figure 3-10. The following observations may help clarify the information given in that figure:

(1) PRIORCULS: Whenever a prior culture is created, it is given the name CULTURE-# (see TYPE), where # is the next unassigned culture number. The values of SITE and WHENCUL are immediately traced using the FINDOUT mechanism (see MAINPROPS). The culture node is put in the context tree below a node of type PERSON (see ASSOCWITH) and the new context may be characterized by clinical parameters of the type PROP-CUL (see PROPTYPE). The prior culture may be the context for either PRCULRULES or CULRULES (see SUBJECT) and is translated, in questions to the user, as "this <site> culture" (see SYN) where "<site>" is replaced by the site of the culture if it is known. The use of PROMPT1

PRIORCULS

ASSOCWITH: PERSON
MAINPROPS: (SITE WHENCUL)
PROMPT1: (Were any organisms that were significant (but no longer require therapeutic attention) isolated within the last approximately 30 days?)
PROMPT2: (Any other significant earlier cultures from which pathogens were isolated?)
PROPTYPE: PROP-CUL
SUBJECT: (PRCULRULES CULRULES)
SYN: (SITE (this * culture))
TRANS: (PRIOR CULTURES OF *)
TYPE: CULTURE-

CURORG

ASSOCWITH: CURCUL
MAINPROPS: (IDENT GRAM MORPH SENSITIVS)
PROMPT2: (Any other organisms isolated from * for which you would like a therapeutic recommendation?)
PROMPT3: (I will refer to the first offending organism from * as:)
PROPTYPE: PROP-ORG
SUBJECT: (ORGRULES CURORGRULES)
SYN: (IDENT (the *))
TRANS: (CURRENT ORGANISMS OF *)
TYPE: ORGANISM-

Figure 3-10: Context trees such as those shown in Figure 3-1 are generated from prototype context-types such as those shown here. The defining "properties" are described in the text.

and PROMPT2 is demonstrated in the sample consultation at the end of Chapter 1.

(2) CURORG: Since there is a PROMPT3 rather than a PROMPT1, MYCIN prints out the PROMPT3 message and assumes (without asking) that there is at least one CURORG for each CURCUL (see ASSOCWITH); the other CURORG properties correspond to those described above for PRIORCULS.

Whenever MYCIN creates a new context using these models, it prints out the name of the new node in the tree, e.g.:

--------ORGANISM-1--------

Thus the user is familiar with MYCIN's internal names for the cultures, organisms, and drugs under discussion. The node names may then be used in MYCIN's questions at times when there may be ambiguity regarding which node is the current context, e.g.:

> Is the patient's illness with the staphylococcus (ORGANISM-2) a hospital-acquired infection?

It should also be noted that when PROMPT1 or PROMPT2 is used to ask the physician a question, he need not be aware that the situation is different from that occurring when FINDOUT asks questions. All the user options described in § 3.3.2-2 operate in the normal fashion.

Finally, the MAINPROPS property requires brief explanation. The claim was previously made that clinical parameters are traced and their values requested by FINDOUT only when they are needed for evaluation of a rule that has been invoked. Yet, we must now acknowledge that certain LABDATA parameters are automatically traced whenever a node for the context tree is created. The reason for this departure is our attempt to keep the program acceptable to physicians. Since the order of rules on UPDATED-BY lists is arbitrary, the order in which questions are asked is somewhat arbitrary as well. We have found that physicians are annoyed if the "basic" questions are not asked first, as soon as the context is created. The MAINPROPS convention forces certain standard questions early in the characterization of a node in the context tree. Parameters not on the MAINPROPS list are then traced in an arbitrary order that depends upon the order in which rules are invoked.

The MAINPROPS convention may be compared to the antecedent theorems of PLANNER that were discussed in § 3.3.1. Although I argued then against a system based solely upon antecedent theorems, I did acknowledge that they were powerful for certain purposes when they did not clutter memory with unnecessary information. Since the parameters on MAINPROPS lists are important pieces of information that would uniformly be traced by FINDOUT anyway, the convention we have implemented forces a standardized ordering of the "basic" questions without generating useless information.

3.4.2 EXPLICIT MECHANISMS FOR BRANCHING

There are two situations under which MYCIN attempts to add new nodes to the context tree. The simpler case occurs when rules explicitly reference contexts that have not yet been created. Suppose, for example, MYCIN is trying to determine the identity of a current organism and therefore invokes the following CURORGRULE:

RULE004

IF: 1) THE IDENTITY OF THE ORGANISM IS NOT KNOWN WITH CERTAINTY, AND
2) THIS CURRENT ORGANISM AND PRIOR ORGANISMS OF THE PATIENT AGREE WITH RESPECT TO THE FOLLOWING PROPERTIES: GRAM MORPH
THEN: THERE IS WEAKLY SUGGESTIVE EVIDENCE THAT EACH OF THEM IS A PRIOR ORGANISM WITH THE SAME IDENTITY AS THIS CURRENT ORGANISM

The second condition in the PREMISE of this rule references other nodes in the tree, namely nodes of the type PRIORORGS. If no such nodes exist, the MONITOR asks FINDOUT to trace PRIORORGS in the normal fashion. The difference is that PRIORORGS is not a clinical parameter but a context-type. FINDOUT therefore uses PROMPT1 of PRIORORGS to ask the user if there is at least one organism. If so, an instantiation of PRIORORGS is added to the context tree and its MAINPROPS are traced. PROMPT2 is then used to see if there are any additional prior organisms and the procedure continues until the user indicates there are no more PRIORORGS that merit discussion. Finally, FINDOUT returns the list of prior organisms to the MONITOR so that the second condition in RULE004 can be evaluated.

3.4.3 IMPLICIT MECHANISMS FOR BRANCHING

There are two kinds of implicit branching mechanisms. One of these is closely associated with the example of the previous section. As shown in Figure 3-1, a prior organism is associated with a prior culture. But the explicit reference to prior organisms in RULE004

made no mention of prior cultures. Thus, if FINDOUT tries to create a PRIORORGS in response to an explicit reference but finds there are no PRIORCULS, the program knows there is an implied need to ask the user about prior cultures before asking about prior organisms. Since PRIORCULS are associated with the patient himself, and since the patient node already exists in the context tree, only one level of implicit branching is required in the evaluation of RULE004.

The other kind of implicit branching occurs when the MONITOR attempts to evaluate a rule for which no appropriate context exists. For example, the first rule invoked in an effort to execute the goal rule is a CURORGRULE (see RULE090, Figure 3-9). Since no current organism has been created at the time the MONITOR is passed this CURORGRULE, MYCIN automatically attempts to create the appropriate nodes and then to apply the invoked rule to each.

3.5 Selection of Therapy

The discussion in § 3.3 and 3.4 concentrated on the PREMISE of MYCIN's principal goal rule (RULE092, § 3.3.2-1). This section explains what happens when the PREMISE is found to be true and the two-step ACTION clause is executed.

Unlike other rules in the system, the goal rule does not lead to a conclusion (§ 3.3.3-2) but instead instigates actions. The functions in the ACTION of the goal rule thus correspond to the <actfunc> class that was introduced in the BNF description of § 3.2.1-2. The first of these functions causes a list of potential therapies to be created. The second allows the best drug or drugs to be selected from the list of possibilities.

3.5.1 CREATION OF POTENTIAL THERAPY LIST

There is a class of decision rules, the THERULES (§ 3.2.2-2), that are never invoked by MYCIN's regular control structure because they do not occur on the UPDATED-BY list of any clinical parameters. These rules contain sensitivity information for the various organisms known to the system. For example:

RULE088

IF: THE IDENTITY OF THE ORGANISM IS PSEUDOMONAS

THEN: I RECOMMEND THERAPY CHOSEN FROM AMONG THE FOLLOWING DRUGS:

1 - COLISTIN	(.98)
2 - POLYMYXIN	(.96)
3 - GENTAMICIN	(.96)
4 - CARBENICILLIN	(.65)
5 - SULFISOXAZOLE	(.64)

The numbers associated with each drug are the probabilities that a pseudomonas isolated at Stanford Hospital will be sensitive (*in vitro*) to the indicated drug. The sensitivity data were acquired from Stanford's microbiology laboratory (and could easily be adjusted to reflect changing resistance patterns at Stanford or the data for some other hospital desiring a version of MYCIN with local sensitivity information). Rules such as the one shown here provide the basis for creating a list of potential therapies. There is one such rule for every kind of organism known to the system.

MYCIN selects drugs only on the basis of the identity of offending organisms. Thus, the program's first task is to decide, for each current organism deemed to be significant, which hypotheses regarding the organism's identity (IDENT) are sufficiently likely so that they must be considered in choosing therapy. MYCIN uses the CF's of the various hypotheses in order to select the most likely identities (see § 4.7). Each identity is then given an "item number" (see below) and the process is repeated for each significant current organism. The "Set of Indications" for therapy is then printed out, e.g.:

My therapy recommendation will be based on the following possible identities of the organism(s) that seem to be significant:

<Item 1> The identity of ORGANISM-1 may be STREPTOCOCCUS-GROUP-D

<Item 2> The identity of ORGANISM-1 may be STREPTOCOCCUS-ALPHA

<Item 3> The identity of ORGANISM-2 is PSEUDOMONAS

Each item in this list of therapy indications corresponds to one of the THERULES. For example, Item 3 corresponds to RULE088 above. Thus, MYCIN retrieves the list of potential therapies for each indication from the associated THERULE. The default (*in vitro*) statistical data are also retrieved. MYCIN then replaces the default sensitivity data with real data about those of the patient's organisms,

if any, for which actual sensitivity information is available from the laboratory. Furthermore, if MYCIN has inferred sensitivity information from the *in vivo* performance of a drug that has already been administered to the patient, this information also replaces the default sensitivity data. Thus, the "compiled list of potential therapies" is actually several lists, one for each item in the Set of Indications. Each list contains the names of drugs and, in addition, the associated number representing MYCIN's judgment regarding the organism's sensitivity to each of the drugs.

3.5.2 SELECTING PREFERRED DRUG FROM LIST

When MYCIN recommends therapy it tries to suggest a drug for each of the items in the Set of Indications. Thus, the problem reduces to one of selecting the best drug from the therapy list associated with each item. Clearly the probability that an organism will be sensitive to a drug is an important factor in this selection process. However, there are several other considerations. MYCIN's strategy is to select the best drug on the basis of sensitivity information but then to consider contraindications for that drug. Only if a drug survives this second screening step is it actually recommended. Furthermore, MYCIN also looks for ways to minimize the number of drugs recommended and thus seeks therapies that cover for more than one of the items in the Set of Indications. The selection/screening process is described in the following two subsections.

3.5.2-1 Choosing Apparent First Choice Drug

The procedure used for selecting the apparent first choice drug is a complex algorithm that is somewhat arbitrary and is currently under revision. In this section, I shall therefore describe the procedure in somewhat general terms since the actual LISP functions and data structures are not particularly enlightening.

There are three initial considerations used in selecting the best therapy for a given item:

(1) the probability that the organism is sensitive to the drug;
(2) whether the drug is already being administered;

(3) the relative efficacy of drugs that are otherwise equally supported by the criteria in (1) and (2).

As is the case with human consultants, MYCIN does not insist on a change in therapy if the physician has already begun a drug that may work, even if that drug would not otherwise be MYCIN's first choice.

Drugs with sensitivity numbers within .05 of one another are considered to be almost identical on the basis of criterion (1). Thus RULE088 above, for example, indicates no clear preference among colistin, polymyxin, and gentamicin for pseudomonas infections (if default sensitivity information from the rule is used). However, our collaborating experts have ranked the relative efficacy of antimicrobials on a scale from 1 to 10. The number reflects such factors as whether the drug is bacteriostatic or bacteriocidal, or its tendency to cause allergic sensitization. Since gentamicin has a higher relative efficacy than either colistin or polymyxin, it is the first drug considered for pseudomonas infections (unless known sensitivity information or previous drug experience indicates that an alternate choice is preferable).

Once MYCIN has selected the apparent best drug for each item in the Set of Indications, it checks to see if one of the drugs is also useful for one or more of the other indications. For example, if the first choice drug for item 1 is the second choice drug for item 2, and if the second choice drug for item 2 is almost as strongly supported as the first choice drug, item 1's first choice drug also becomes item 2's first choice drug. This strategy permits MYCIN to attempt to minimize the number of drugs to be recommended.

A similar strategy is used to avoid giving two drugs of the same drug class. For example, MYCIN knows that if the first choice for one item is penicillin and the first choice for another is ampicillin, then the ampicillin may be given for both indications.

In the ideal case, MYCIN will find a single drug that effectively covers for all the items in the set of indications. But even if each item remains associated with a different drug, a screening stage to look for contraindications is required. This rule-based process is described in the next subsection. It should be stressed, however, that the manipulation of drug lists described above is algorithmic, i.e., it is coded in LISP functions that are called from the ACTION clause of the goal rule. There is considerable "knowledge" in this process. Since rule-

based knowledge provides the foundation of MYCIN's ability to explain its decisions, it would be desirable eventually to remove this therapy selection method from functions and place it in decision rules. I will return to this point in § 3.7.

3.5.2-2 Rule-based Screening for Contraindications

Unlike the complex list manipulations described in the previous subsection, criteria for ruling out drugs under consideration may be effectively placed in rules. The rules in MYCIN for this purpose are termed ORDERRULES. The advantages to placing this knowledge in rules are the ones I discussed in Chapter 2, i.e., modularity, ease of modification, and facilitation of explanation and other question–answering. A sample rule of this type is:

RULE055

IF: 1) THE THERAPY UNDER CONSIDERATION IS TETRACYCLINE, AND
2) THE AGE (IN YEARS) OF THE PATIENT IS LESS THAN 13
THEN: THERE IS STRONGLY SUGGESTIVE EVIDENCE (.8) THAT TETRACYCLINE IS NOT AN APPROPRIATE THERAPY FOR USE AGAINST THE ORGANISM

In order to use MONITOR and FINDOUT with such rules, we must construct appropriate nodes in the context tree and must be able to characterize them with clinical parameters. The context-type used for this purpose is termed POSSTHER (§ 3.2.2-1) and the parameters are classified as PROP-THER (§ 3.2.3-2). Thus, when MYCIN has selected the apparent best drugs for the items in the Set of Indications, it creates a context corresponding to each of these drugs. POSSTHER contexts occur below CURORGS in the context tree. FINDOUT is then called to trace the relevant clinical parameter that collects contraindication information (i.e., this becomes a new goal statement) and the normal recursive mechanism through the MONITOR insures that the proper ORDERRULES are invoked.

ORDERRULES allow a great deal of drug-specific knowledge to be stored. For example, RULE055 above insures that tetracycline is ruled out in youngsters who still have developing bone and teeth.

Similar rules tell MYCIN never to given streptomycin or carbenicillin alone, not to give sulfonamides except in urinary tract infections, and not to give cephalothin, clindamycin, lincomycin, vancomycin, cefazolin, or erythromycin if the patient has meningitis. Other ORDERRULES allow MYCIN to consider the patient's drug allergies, dosage modifications, or ecological considerations (e.g., save gentamicin for pseudomonas, serratia, and hafnia unless the patient is so sick that you cannot risk using a different aminoglycoside while awaiting lab sensitivity data). Finally, there are rules that suggest appropriate combination therapies (e.g., add carbenicillin to gentamicin for known pseudomonas infections). In considering such rules MYCIN often is forced to ask questions that never arose during the initial portion of the consultation. Thus the physician is asked additional questions during the period after MYCIN has displayed the items in the Set of Indications but before any therapy is actually recommended.

After the presumed first-choice drugs have been exposed to the ORDERRULE screening process, MYCIN checks to see whether any of the drugs is now contraindicated. If so, the process described in § 3.5.2-1 is repeated. New first-choice drugs are then subjected to the ORDERRULES as I have described above. The process continues until all the first-choice drugs are found to have been instantiated already as POSSTHERS. These then become the system's recommendations. Note that this strategy may result in the recommendation of drugs that are only mildly contraindicated so long as they are otherwise strongly favored. The therapy recommendation itself takes the following form:

My preferred therapy recommendation is as follows:
In order to cover for Items <1> <2> <3>:
Give the following in combination:
1. PENICILLIN
   Dose: 285,000 UNITS/KG/DAY - IV
2. GENTAMICIN
   Dose: 1.7 MG/KG Q8H - IV OR IM
   Comments: MODIFY DOSE IN RENAL FAILURE

The user may also ask for second, third, and subsequent therapy recommendations until MYCIN is able to suggest no reasonable alternatives. The mechanism for these iterations is merely a repeat of

the processes described above but with recommended drugs removed from consideration.

3.6 Mechanisms for Storage of Patient Data

3.6.1 CHANGING ANSWERS TO QUESTIONS

If a physician decides he wants to change his response to a question that he has already answered, MYCIN must do more than merely redisplay the prompt, accept the user's new answer, and make the appropriate change to the value of the clinical parameter in question. The question was originally asked because the PREMISE of a decision rule referenced the clinical parameter. Thus, his original response affected the evaluation of at least one rule, and subsequent pathways in the reasoning network may have been affected as well. It is therefore necessary for MYCIN somehow to return to the state it was in at the time the question was originally asked. Its subsequent actions can then be determined by the corrected user response.

Reversing all decisions made since a question was asked is a complex problem, however. The most difficult task is to determine what portions of a parameter's cumulative CF preceded or followed the question requiring alteration. In fact, the extra data structures needed to permit this kind of backing up are so large and complicated, and would be used so seldom, that it seems preferable simply to restart the consultation from the beginning when the user wants to change one of his answers.

Restarting is of course also less than optimal, particularly if it requires that the physician reenter the answers to questions that were correct the first time around. Our desire to make the program acceptable to physicians required that we devise some mechanism for changing answers, but restarting from scratch also had obvious drawbacks regarding user acceptance of the system. We therefore needed a mechanism for restarting MYCIN's reasoning process but avoiding questions that had already been answered correctly. When FINDOUT asks questions it therefore uses the following algorithm:

[1] before asking the question, check to see if the answer is already stored (in the Patient Data Table, see [3]); if the answer is there, use that value rather than asking the user; otherwise go to [2].

[2] ask the question using PROMPT or PROMPT1 as usual.
[3] store the user's response in the Patient Data Table under the appropriate clinical parameter and context.

The Patient Data Table, then, is a growing record of the user's responses to questions from MYCIN (see Patient Data, Figure 1-1). It is entirely separate from the dynamic data record (§ 3.3.3-1) that is explicitly associated with the nodes in the context tree. Note that the Patient Data Table contains only the text responses of the user; there is no CF information (unless included in the user's response), nor are there data derived from MYCIN's rule-based inferences.

The Patient Data Table and the FINDOUT algorithm above make the task of changing answers much simpler. The technique MYCIN uses is the following:

[a] Whenever the user wants to change the answer to a previous question, he enters CHANGE <numbers>, where <numbers> is a list of the questions whose answers need correction (see § 3.3.2-2);
[b] MYCIN looks up the indicated question numbers in its question record (see § 3.3.3-3);
[c] The user's responses to the indicated questions are removed from the current Patient Data Table;
[d] MYCIN reinitializes the system, erasing the entire context tree, including all associated parameters; however, it leaves the Patient Data Table intact except for the responses deleted in [c];
[e] MYCIN restarts the consultation from the beginning.

This simple mechanism results in a restarting of the Consultation System (Subprogram 1) but does not require that the user enter correct answers a second time. Since the Patient Data Table is saved, step [1] of the FINDOUT algorithm above will find all the user's responses until the first question requiring alteration is reached. Thus, the first question asked the user after he gives the CHANGE command is, in fact, the earliest of the questions he wants to change. There may be a substantial pause after the CHANGE command while MYCIN reasons through the network to the first question requiring alteration, but a pause is to be preferred over a mechanism requiring reentry of all question answers. The implemented technique is entirely general because answers to questions regarding tree propagation (§ 3.4.1) are also stored in the Patient Data Table.

3.6.2 REMEMBERING PATIENTS FOR FUTURE REFERENCE

When a consultation is complete, the Patient Data Table contains all responses necessary for generating a complete consultation for that patient. It is therefore straightforward to store the Patient Data Table (on disk or tape) so that it may be reloaded in the future. FINDOUT will automatically read responses from the Table, rather than ask the user, so a consultation may be run several times on the basis of only a single interactive session.

There are two reasons for storing Patient Data Tables for future reference. One is their usefulness in evaluating changes to MYCIN's knowledge base. The other is the resulting ability to re-evaluate patients once new clinical information becomes available.

3.6.2-1 Evaluating New Rules

New rules may have a large effect on the way a given patient case is handled by MYCIN. For example, a single rule may reference a clinical parameter not previously sought or may lead to an entirely new chain in the reasoning network. It is therefore useful to reload Patient Data Tables and run a new version of MYCIN on old patient cases. A few new questions may be asked (because their responses are not stored in the Patient Data Table). Conclusions regarding organism identities may then be observed, as may the program's therapeutic recommendations. Any changes from the decisions reached during the original run (i.e., when the Patient Data Table was created) must be explained. When a new version of MYCIN evaluates several old Patient Data Tables in this manner, aberrant side effects of new rules may be found. Thus stored patient cases provide a useful mechanism for screening new rules before they become an integral part of MYCIN's knowledge base.

3.6.2-2 Re-evaluating Patient Cases

The second use for stored Patient Data Tables is the re-evaluation of a patient once additional laboratory or clinical information becomes available. If a user answers several questions with UNKNOWN during the initial consultation session, MYCIN's advice will, of course, be based on less than complete information. After storing the

Patient Data Table, however, the physician may return for another consultation in a day or so once he has more specific information. MYCIN can use the previous Patient Data Table for responses to questions whose answers are still up-to-date. The user therefore needs to answer only those questions that reference new information. A mechanism for the physician to indicate directly what new data are available has not yet been automated, however.

A related capability to be implemented before MYCIN becomes available in the clinical setting is a SAVE command. If a physician must leave the computer terminal midway through a consultation, this option will save the current Patient Data Table on the disk. When he returns to complete the consultation he will reload the patient record and the session will continue from the point at which he entered the SAVE command.

It should be understood that saving the current Patient Data Table is *not* the same as saving the current state of MYCIN's reasoning. Thus, as we have stated above, changes to MYCIN's rule corpus may result in different advice from an identical Patient Data Table. Finally, I wish to emphasize our awareness that disk storage of patient information immediately raises questions of data confidentiality. We will attempt to insure appropriate data protection when MYCIN is available in the clinical setting.

3.7 Future Extensions

In this section I summarize some current ideas for improvement of the consultation program described in this chapter. Each of the topics mentioned is the subject of current efforts by one or more of the researchers currently associated with the MYCIN project.

3.7.1 DYNAMIC ORDERING OF RULES

The order in which rules are invoked by the MONITOR is currently controlled solely by their order on the UPDATED-BY property of the clinical parameter being traced. (An exception to this point is the self-referencing rules, see § 3.3.4.) The order of rules on the UPDATED-BY property is also arbitrary, tending to reflect nothing more than the order in which rules were acquired. Since FINDOUT

sends all rules on such lists to the MONITOR, and since our certainty factor combining function is commutative (§ 4.6), the order of rules is unimportant.

Some rules are much more useful than others in tracing the value of a clinical parameter. For example, a rule with a six-condition PREMISE that infers the value of a parameter with a low CF requires a great deal of work (as many as six calls to FINDOUT) with very little gain. On the other hand, a rule with a large CF and only one or two PREMISE conditions may easily provide strong evidence regarding the value of the parameter in question. It may therefore be wise for FINDOUT to order the rules in the UPDATED-BY list on the basis of both information content (CF) and the work necessary to evaluate the PREMISE. Then if the first few rules are successfully executed by the MONITOR, the CF associated with one of the values of the clinical parameter may be so large that invocation of subsequent rules will require more computational effort than they are worth. If FINDOUT therefore ignores such rules (i.e., does not bother to pass them to the MONITOR), considerable time savings may result. Furthermore, entire reasoning chains will in some cases be avoided and the number of questions asked the user could accordingly be decreased.

3.7.2 DYNAMIC ORDERING OF CONDITIONS WITHIN RULES

The MONITOR diagram in Figure 3-7 reveals that conditions are evaluated strictly in the order that they occur within the PREMISE of the rule. In fact, I have stressed that the order of conditions is therefore important and that the most commonly referenced clinical parameters should be placed earliest in the PREMISE.

Suppose, however, that in a given consultation the clinical parameter referenced in the fourth condition of a rule has already been traced by FINDOUT because it was referenced in some other rule that the MONITOR has already evaluated. As currently designed, MYCIN checks the first three conditions first, even if the fourth condition is already known to be false. Since the first three conditions may well require calls to FINDOUT, the rule may generate unnecessary questions and expand useless reasoning chains.

The solution to this problem would be to redesign the MONITOR so that it reorders the PREMISE conditions, first evaluating those

that reference clinical parameters which have already been traced by FINDOUT. In this way a rule will not cause new questions nor additions to the reasoning network if any of its conditions are known to be false at the outset.

3.7.3 PRE-SCREENING OF RULES

An alternate approach to the problem described in the preceding section would be for FINDOUT to judge the implications of every parameter it traces. Once the value has been determined by the normal mechanism, FINDOUT could use the LOOKAHEAD list for the clinical parameter in order to identify all rules referencing the parameter in their PREMISE conditions. FINDOUT could then evaluate the relevant conditions and mark the rule as failing if the condition turns out to be false. Then, whenever the MONITOR begins to evaluate rules that are invoked by the normal recursive mechanism, it will check to see if the rule has previously been marked as false by FINDOUT. If so, the rule could be quickly ruled out without needing to consider the problem of re-ordering the PREMISE conditions.

At first glance, the dynamic re-ordering of PREMISE conditions appears to be a better solution than the one I have just described. The problem with rule pre-screening is that it requires consideration of all rules on the parameter's LOOKAHEAD list, some of which may never actually be invoked during the consultation. Thus the disadvantages are similar to those that can accompany the PLANNER antecedent theorems that were previously described (§ 3.3.1).

3.7.4 PLACING ALL KNOWLEDGE IN RULES

Although most of MYCIN's knowledge is placed in decision rules, I have pointed out several examples of knowledge that is not rule-based. The simple lists and knowledge tables of § 3.2.6 may perhaps be justified on the basis of efficiency arguments, especially since those knowledge structures may be directly accessed and utilized by rules.

However, the algorithmic mechanisms for therapy selection that were described in § 3.5 are somewhat more bothersome. Although we have managed to put many drug-related decision criteria in the

ORDERRULES, the mechanisms for creating the potential therapy lists and for choosing the apparent first choice drug are programmed explicitly in a series of relatively complex LISP functions. Since MYCIN's ability to explain itself is based upon rule-retrieval (Chapter 5), the system cannot give good descriptions of these drug selection procedures. It is therefore desirable to place more of the drug selection knowledge in rules.

Such efforts should provide a useful basis for evaluating the power of our rule-based formalism. If the goal-oriented control structure we have developed is truly general, one would hope that algorithmic approaches to the construction and ordering of lists could also be placed in decision rule format. We therefore intend to experiment with ways of incorporating the remainder of MYCIN's knowledge into decision rules that are invoked by the standard MONITOR/FINDOUT process.

3.7.5 NEED FOR CONTEXT GRAPH

The context tree used by MYCIN is the source of one of the system's primary problems in attempting to simulate the consultation process. As was pointed out in § 3.2.2-1, every node in the context tree leads to the uppermost patient node by a single pathway. In reality, however, drugs, patients, organisms, and cultures are not interrelated in this highly structured fashion. For example, drugs are often given to cover for more than one organism. The context tree does not permit a single CURDRUG or PRIORDRUG to be associated with more than a single organism. What we need, therefore, is a network of contexts in the form of a graph rather than a pure tree. The current reasons why MYCIN needs a tree structured context network are explained in § 3.2.2. We have come to recognize that a context graph capability is an important extension of the current system, however, and this will be the subject of future design modifications. When implemented, for example, it will permit a physician to discuss a prior drug only once even though it may have been given to cover for several prior organisms.

3.8 Advantages of MYCIN Approach

There are four principal advantages of the MYCIN approach that have contributed to the system's current level of success. Each of

these distinguishes MYCIN from the medical decision making programs described in § 1.3. They also reflect MYCIN's debt to previous work in the AI field.

3.8.1 MODULARITY OF KNOWLEDGE

As discussed in Chapter 2, one of the major design considerations during the development of MYCIN has been the isolation of pieces of knowledge as discrete facts. MYCIN's decision rules achieve this goal. Since each rule represents a discrete packet of knowledge, the integration of new information into the system is simplified. Furthermore, the rules can serve as the basis for MYCIN's explanation and question-answering capabilities (Chapter 5).

Modularity of knowledge is seldom found in diagnostic programs. Some statisticians would argue, in fact, that the interrelationships of observations are so complex that a formal Bayesian approach is the only reasonable way to guarantee good predictions. As I argue in Chapter 4, however, the statistician's stance is greatly weakened when the knowledge is primarily judgmental and it defies statistical formulation. By accepting the inexact nature of many medical decisions, and by acknowledging that the quantification scheme accompanying our rules is only an approximation technique, we are left free to isolate our knowledge statements and to reap the associated benefits provided by that representation scheme. In fact, almost all of those capabilities that make MYCIN truly innovative may be directly attributed to the program's rule-based representation of knowledge.

3.8.2 DYNAMIC REASONING CHAIN

It is reasonable to ask why MYCIN does not create an explicit decision tree from its rules, code them for maximal efficiency, and then rely upon conventional techniques for decision analysis based upon progression through a branching tree. It must be remembered, however, that the reasoning network for MYCIN is goal-oriented (Figure 3-9). Conventional decision trees start at the top node and follow a path through the tree based on decisions reached at each subsequent node. When a terminal node in the tree is finally reached, that is the diagnosis. MYCIN's terminal nodes, on the other hand, correspond to starting points in the accumulation of data (i.e., ASK1

or ASK2 nodes, Figure 3-9). MYCIN's task is to determine which of these terminal nodes to use in an effort to reach the top of the tree. Thus, the form of MYCIN's reasoning network is distinctive from a conventional decision tree in that the top node represents the goal for MYCIN rather than the starting point.

Although MYCIN's rules do not naturally form a conventional decision tree, it is possible that a researcher with experience constructing decision trees could, in time, convert MYCIN's knowledge base into a traditional tree-shaped format. This has not seemed to be a particularly natural approach, however. There are three principal factors that would complicate any such attempt:

(1) Although decision theory has provided mechanisms for incorporating probabilistic knowledge into decision trees, there is no obvious mechanism for combining MYCIN's certainty factors with a branching network;
(2) MYCIN's current control structure depends upon a dynamic set of contexts and the ability to use rules more than once; this suggests that a decision tree using MYCIN's knowledge would need to have mechanisms for reusing certain portions, perhaps by defining decision tree "macros";
(3) MYCIN's reasoning network is actually not tree-shaped; this complexity was not shown in the sample network of Figure 3-9, but since MYCIN's rules often form reasoning loops (§ 3.3.5) and since a single observation often affects several of the ascending branches in the network, a graph structure would actually provide a more accurate representation of MYCIN's reasoning network.

It has also been suggested that, even if we do not convert MYCIN's reasoning network to a conventional decision tree, we could at least explicitly "compile" it. It should be noted, however, that since MYCIN works backwards from the goal-rule, there is no disadvantage to creating a dynamic reasoning chain as it proceeds. The total network that could be created from MYCIN's rules is so vast that it appears preferable simply to create the portion of the network that is appropriate for the patient under consideration. An explicit network would not avoid the need for MYCIN to work backwards from the topmost goal node. Furthermore, it would introduce the obvious disadvantage that newly acquired rules could not be automatically incorporated into MYCIN's reasoning as they are by the current dynamic control structure.

3.8.3 DOMAIN-INDEPENDENT CONTROL STRUCTURE

Except for the functions described in § 3.5, most of MYCIN's functions are domain-independent. In particular, the entire MONITOR/FINDOUT mechanism contains no explicit knowledge of the problem domain for which it has been designed. It is therefore tempting to consider writing new rules for additional medical (or nonmedical) problem areas and to see whether the MYCIN formalism will allow valid consultations in those areas as well. Of course, new clinical parameters and their associated properties would also have to be created, but the resulting knowledge structures are designed to be capable of forming the basis both for consultation sessions using Subprogram 1 and for question-answering sessions using Subprogram 2 (Chapter 5).

Use of the MYCIN approach for another problem area has not yet been attempted, however, and it would therefore be premature to claim that MYCIN's approach can indeed be generalized for other domains. One reason that we have not attempted to apply the approach elsewhere is our recognition, based on experience to date, that the formulation of new decision rules is no straightforward matter, at least for medical applications. Physicians have not in general structured their own decision processes, and a clinical expert who consistently makes excellent recommendations may have great difficulty describing the steps in reasoning that he uses to make his decisions. Thus, although we are hopeful that the MYCIN formalism can be adapted to another problem area with minimal modification, such efforts would be distracting at a time when our principal concern is the expansion of MYCIN's clinical expertise regarding antimicrobial therapy.

3.8.4 REASONING WITH JUDGMENTAL KNOWLEDGE

The primary advantage of the MYCIN approach, however, is its ability to model medical reasoning that is based upon neither diagnostic algorithms, physiological models, nor statistical analysis. In fact, MYCIN's principal contribution to the field of computer-based medical decision making may well be its reasoning model that uses the informal judgmental knowledge of physician experts. Other programs have attempted to use "estimates" provided by expert physi-

cians [Leaper, 1972] but have been limited by efforts to couch these estimates in probabilistic terms. MYCIN not only provides an intuitively pleasing mechanism for recording (decision rules) and interpreting (certainty factors) these numbers, but it provides a flexible control structure and interactive capabilities that encourage the physician to accept the program as the useful and cooperative clinical tool that it is designed to be.