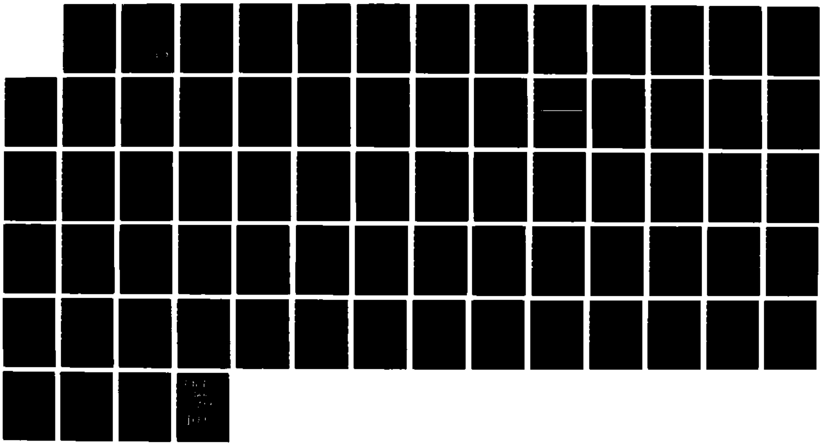
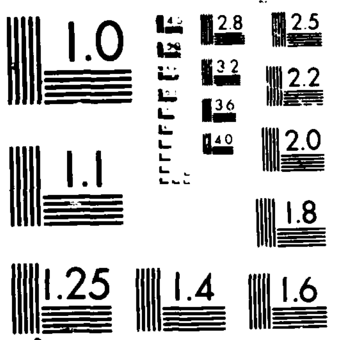
AD-A107 021 REPRESENTING CONTROL KNOWLEDGE AS ABSTRACT TASK AND METARULES(U) STANFORD UNIV CA KNOWLEDGE SYSTEMS LAB W J CLANCEY ET AL. APR 85 KSL-85-16 N00014-85-K-0305

1/1

UNCLASSIFIED F/G 12/7 NL




1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

April 1985

Report No. STAN-CS-87-1168
Also numbered KSL-85-16

4

Representing Control Knowledge as Abstract Task and Metarules

by

William J. Clancey and Conrad Bock

Department of Computer Science

Stanford University
Stanford, CA 94305

87

AD-A187021

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT: APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S): ONR TECHNICAL REPORT #15
6a NAME OF PERFORMING ORGANIZATION: STANFORD KNOWLEDGE SYSTEMS LABORATORY; 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION: PERSONNEL AND TRAINING RESEARCH PROGRAMS
6c. ADDRESS (City, State, and ZIP Code): COMPUTER SCIENCE DEPARTMENT, 701 WELCH ROAD, BUILDING C, PALO ALTO, CA 94304	7b ADDRESS (City, State, and ZIP Code): OFFICE OF NAVAL RESEARCH (CODE 1142PT), 800 NORTH QUINCY STREET, ARLINGTON, VA 22217-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N00014-85K-0305
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO
61153N	RR04206	0C	NR702-003

11 TITLE (Include Security Classification)
REPRESENTING CONTROL KNOWLEDGE AS ABSTRACT TASK AND METARULES

12 PERSONAL AUTHOR(S)
WILLIAM J. CLANCEY & CONRAD BOCK

13a TYPE OF REPORT: TECHNICAL	13b TIME COVERED FROM ____ TO ____	14 DATE OF REPORT (Year, Month, Day): APRIL 1985	15 PAGE COUNT: 68

16 SUPPLEMENTARY NOTATION
KSL 85-16 TO APPEAR IN COMPUTER EXPERT SYSTEMS, M. COOMBS AND L. BLOC (eds.), SPRINGER VERLAG, IN PRESS

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	CONTROL KNOWLEDGE, METAKNOWLEDGE, STRATEGY, PROCEDURAL KNOWLEDGE, EXPLANATION, LOGIC
05	09		

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

A poorly designed knowledge base can be as cryptic as an arbitrary program and just as difficult to maintain. Representing inference procedures abstractly, separately from domain facts and relations, makes the design more transparent and explainable. The combination of abstract procedures and a relational language for organizing domain knowledge provides a generic framework for constructing knowledge bases for related problems in other domains and also provides a useful starting point for studying the nature of strategies. In HERACLES inference procedures are represented as abstract metarules, expressed in a form of the predicate calculus, organized and controlled as rule sets. A compiler converts the rules into Lisp code and allows domain relations to be encoded as arbitrary data structures for efficiency. Examples are given of the explanation and teaching capabilities afforded by this representation. Different perspectives for understanding HERACLES' inference procedure and how it defines a relational knowledge base are discussed in some detail.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT: [✓ UNCLASSIFIED/UNLIMITED] [✓ SAME AS RPT] ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL: DR. SUSAN CHIPMAN	22b TELEPHONE (Include Area Code): (202) 696-4318; 22c OFFICE SYMBOL: ONR 1142PT

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Representing Control Knowledge as Abstract Task and Metarules

by

William J. Clancey

&

Conrad Bock

Stanford Knowledge Systems Laboratory
Department of Computer Science
701 Welch Road, Building C
Palo Alto, CA 94304

The studies reported here were supported (in part) by:

The Office of Naval Research
Personnel and Training Research Programs
Psychological Sciences Division
Contract No. N00014-85K-0305

The Josiah Macy, Jr. Foundation
Grant No. B852005
New York City

Table of Contents

List of Figures

List of Tables

Abstract

A poorly designed knowledge base can be as cryptic as an arbitrary program and just as difficult to maintain. Representing inference procedures abstractly, separately from domain facts and relations, makes the design more transparent and explainable. The combination of abstract procedures and a relational language for organizing domain knowledge provides a generic framework for constructing knowledge bases for related problems in other domains and also provides a useful starting point for studying the nature of strategies. In HERACLES inference procedures are represented as abstract metarules, expressed in a form of the predicate calculus, organized and controlled as rule sets. A compiler converts the rules into Lisp code and allows domain relations to be encoded as arbitrary data structures for efficiency. Examples are given of the explanation and teaching capabilities afforded by this representation. Different perspectives for understanding HERACLES' inference procedure and how it defines a relational knowledge base are discussed in some detail.

1. Introduction

A important feature of knowledge-based programs, distinguishing them from traditional programs, is that they contain well-structured statements of *what is true about the world* that are separate from *what to do to solve problems*. At least in principle, this separation makes it possible to write programs that interpret knowledge bases from multiple perspectives, providing the foundation for explanation, learning, and teaching capabilities (Davis, 1976, Szolovits, et al., 1978, de Kleer, 1979, Swartout, 1981, Moore, 1982, Clancey, 1983, Genesereth, 1983). The basic considerations in realizing this design are:

1. *Abstraction:* To enable multiple use, inference procedures should be stated separately, not instantiated and composed with the domain facts they manipulate;

2. *Interpretability:* Both factual and procedural knowledge should be stated in a language that multiple programs can interpret (including a natural language translator), incorporating levels of abstraction that facilitate manipulation (an issue of *perspicuity*);

3. *Rationale:* Underlying constraints that justify the design of procedures and models of the world supporting domain facts may be useful for explanation as well as problem solving.

In the GUIDON program (Clancey, 1979, Clancey, 1982a) we explored the advantages and limitations of MYCIN's simple, rule-based architecture as a knowledge representation for a teaching program. To resolve some of the difficulties, we devised a new architecture for a program called NEOMYCIN. In NEOMYCIN (Clancey, 1981, Clancey, 1984a, Clancey, 1984b), the medical (domain) knowledge and diagnostic procedure of MYCIN are expanded (to provide more material for teaching) and represented separately and explicitly. Figure 1-1 shows this idea in a simple way. We also refer to the diagnostic procedure as *strategic* or *control knowledge*. The procedure indexes the domain knowledge, deciding what information to gather about a problem and what assertions to make. The representation of this procedure in a manner that facilitates explanation, student modeling, and ease of maintenance is the main subject of this paper. The complexity of the diagnostic procedure, its abstract nature, and the requirements of the teaching application for interpretability and explicit rationale distinguish this research from previous work.

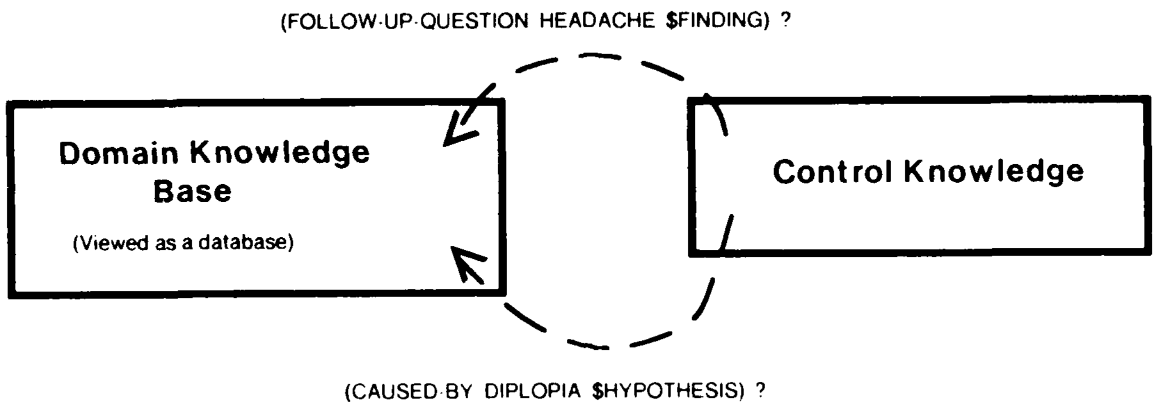

(FOLLOW-UP-QUESTION HEADACHE $FINDING) ?
Domain Knowledge Base
(Viewed as a database)
Control Knowledge
(CAUSED-BY DIPLOPIA $HYPOTHESIS) ?

Figure 1-1: Separation of domain and control knowledge in NEOMYCIN

The development of a good procedural language can be viewed from several perspectives, reflecting the evolutionary process of finding a good representation, using it, and generalizing:

1. *Specify*. We studied MYCIN's rules and identified two recurrent, implicit strategies: a hypothesis refinement strategy (Section 2.1) and a question-asking strategy (Section 2.2). We significantly augmented the procedure to incorporate a more complete model of human diagnostic reasoning that is useful for teaching (Section 3).

2. *Represent*. To meet our design criteria, we chose to represent the control knowledge as sets of rules, organized into subprocedures called *tasks*. To make explicit how domain knowledge is used by the control rules, we chose a predicate calculus language coupled with procedural attachment (Section 4). This combination makes explicit the relations among concepts, allows variables for generality, and allows arbitrary representations of domain knowledge for efficiency.

3. *Re-represent*. Observing that the interpreter for the control knowledge was still implicit in Lisp code, we re-implemented it as a simple rule-based system (Section 6). The experiment failed: Expressing knowledge in rules and predicate calculus does not mean that the notation is readable by people or easily interpreted for multiple purposes. A notation is not inherently *declarative*; instead, this is a relation between *a notation* and *an interpreter that decodes the notation for some purpose* (Rumelhart and Norman, 1983). The nature of the knowledge to be

decoded, and thus the *expressibility* of a given notation, may change with the purpose.

4. *Make practical.* To make the control knowledge interpretation process more efficient, a compiler was written to compile control rules into Lisp code, replacing relational expressions by direct access to domain data structures (Section 5).

5. *Exploit.* Given a "rough draft" notation that brings about appropriate problem solving performance, we developed explanation and student modeling programs to demonstrate the adequacy of the notation for meeting our design goals. We discovered that some additional knowledge, not required for problem solving, is useful for interpreting the control knowledge for other purposes (Section 7).

6. *Generalize.* In studying NEOMYCIN, we determined that the control knowledge is a general procedure for *heuristic classification* (Clancey, 1985). In comparing NEOMYCIN to other programs, we determined that it does diagnosis by selecting a *system identification* from a taxonomy pre-enumerated in the knowledge base. Thus, the program's architecture embodies a general problem solving method for constructing or interpreting arbitrary systems by selecting from pre-enumerated solutions.

   Extracting the domain knowledge from NEOMYCIN, as was done in creating EMYCIN *from MYCIN, we named the framework* HERACLES, "Heuristic Classification Shell," consisting of the classification procedure and interpreter, the relational language for stating domain knowledge and procedural attachments, compiler, and explanation program. We used the HERACLES framework to construct another, non-medical knowledge system in the domain of cast iron defects diagnosis (Section 8.1). We demonstrate the generality of the task language by using it to state the explanation program (Section 8.2).

7. *Study.* We studied the collected body of control knowledge and discovered patterns revealing how the meaning of a procedure is tied up in the relational classification of domain knowledge (Section 9). We found that the predicate calculus notation is extremely valuable for revealing these patterns. We examined the difficulties of achieving the ideal separation between domain and control knowledge to characterize situations in which it is impractical or impossible.

The body of this paper unfolds the development of NEOMYCIN, HERACLES, explanation, modeling, and application programs, as indicated above. The central theme is that *an important design principle for building knowledge-based systems is to represent all control knowledge abstractly, separate from the domain knowledge it operates upon.* In essence, we are applying the familiar principle of separating programs from data, but in the context of knowledge-based programming. We argue that the advantages for construction and maintenance of such programs are so pronounced that benefits will accrue from using this approach, even if there is no interest in using the knowledge base for explanation or teaching. The many scientific, engineering, and practical benefits are summarized in Section 11. This work is extensively compared to other research in Section 10.

2. What is abstract control knowledge?

We begin with a simple introduction to the idea of abstract control knowledge and examples of alternative representations. "Control knowledge" specifies when and how a program is to carry out its operations, such as pursuing a goal, focusing, acquiring data, and making inferences. A basic distinction can be made between the facts and relations of a knowledge base and the program operations that act upon it. For example, facts and relations in a medical knowledge base might include (expressed in a predicate calculus formulation):

```
(SUBTYPE INFECTION MENINGITIS)
   -- "meningitis is a kind of infection"

(CAUSES INFECTION FEVER)
   -- "infection causes fever"

(CAUSES INFECTION SHAKING-CHILLS)
   -- "infection causes shaking chills"

(DISORDER MENINGITIS)
   -- "meningitis is a disorder"

(FINDING FEVER)
   -- "fever is a finding"
```

Such a knowledge base might be used to provide consultative advice to a user, in a way typical of expert systems (Duda and Shortliffe, 1983). Consider, for example, a consultation system for diagnosing some faulty device. One typical program operation is to select a finding that causes a disorder and ask the user to indicate whether the device being diagnosed exhibits that symptom. Specifically, a medical diagnostic system might ask the user whether the patient is suffering from shaking chills, in order to determine whether he has an infection. The first description of the program's operation is *abstract*, referring only to domain-independent relations like "finding" and "causes"; the second description is *concrete*, referring to domain-dependent terms like "shaking-chills" and "infection". ("Domain-independent" doesn't mean that it applies to every domain, just that the term is not specific to any one domain.)

The operation described here can be characterized abstractly as "attempting to confirm a diagnostic hypothesis" or concretely as "attempting to determine whether the patient has an infection." Either description indicates the *strategy* that motivates the question the program is asking of the user. So in this example we see how a strategy, or control knowledge, can be stated either abstractly or concretely. The following two examples illustrate how both forms of control knowledge might be represented in a knowledge base.

2.1. An implicit refinement strategy

In MYCIN (Shortliffe, 1976), most knowledge is represented as domain-specific rules. For example, the rule "If the patient has an infection and his CSF cell count is less than 10, then it is unlikely that he has meningitis," might be represented as:

```
PREMISE:
   ($AND (SAME CNTXT INFECTION)
         (ILESSP (VAL1 CNTXT CSFCELLCOUNT) 10))
ACTION:
   (CONCLUDE CNTXT INFECTION-TYPE MENINGITIS TALLY -700)
```

The order of clauses is important here, for the program should not consider the "CSF cell count" if the patient does not have an infection. Such clause ordering in all rules ensures that the program proceeds by top-down refinement from infection to meningitis to subtypes of meningitis. The disease hierarchy cannot be stated explicitly in the MYCIN rule language; it is implicit in the design of the rules. (See (Clancey, 1983) for further analysis of the limitations of MYCIN's representation.)

CENTAUR (Aikins, 1980), derived from MYCIN, is a system in which disease hierarchies are explicit. In its representation language, MYCIN's meningitis knowledge might be encoded as follows (using a Lisp property list notation):

```
INFECTION
   MORE-SPECIFIC    ((disease MENINGITIS)
                     (disease BACTEREMIA)...)
   IF-CONFIRMED     (DETERMINE disease of INFECTION)

MENINGITIS
   MORE-SPECIFIC    ((subtype BACTERIAL)
                     (subtype VIRAL)...)
   IF-CONFIRMED     (DETERMINE subtype of MENINGITIS)
```

In CENTAUR, hierarchical relations among disorders are explicit (meningitis is a specific kind of infection), and the strategies for using the knowledge are domain-specific (after confirming that the patient has an infection, determine what more specific disease he has). This design enables CENTAUR to articulate its operations better than MYCIN, whose hierarchical relations and strategy are procedurally embedded in rules.

However, observe that each node of CENTAUR's hierarchy essentially repeats a single strategy--try to confirm the presence of a child disorder--and the overall strategy of top-down refinement is not explicit. Aikins has *labeled* CENTAUR's strategies, but has not stated them abstractly. By representing strategies abstractly, it is possible to have a more explicit and non-redundant design. This is what is done in NEOMYCIN.

In NEOMYCIN domain relations and strategy are represented *separately* and strategy is represented abstractly. A typical rule that accomplishes, in part, the abstract task of attempting to confirm a diagnostic hypothesis and its subtypes is shown below.

<Domain Knowledge>

```
INFECTION
   CAUSAL-SUBTYPES   (MENINGITIS BACTEREMIA ...)

MENINGITIS
   CAUSAL-SUBTYPES   (BACTERIAL VIRAL ...)
```

<Abstract Control Knowledge>

```
TASK: EXPLORE-AND-REFINE
ARGUMENT: CURRENT-HYPOTHESIS

METARULE001
IF the hypothesis being focused upon
        has a child
        that has not been pursued,
THEN pursue that child.

(IF (AND (CURRENT-ARGUMENT $CURFOCUS)
         (CHILDOF $CURFOCUS $CHILD)
         (THNOT (PURSUED $CHILD)))
    (NEXTACTION (PURSUE-HYPOTHESIS $CHILD)))
```

NEOMYCIN uses a deliberation/action loop for deducing what it should do next. *Metarules*, like the one shown above, recommend what task should be done next, what domain rule applied, or what domain finding requested from the user (details are given in Section 4.1). The important thing to notice is that this metarule will be applied for refining any disorder, obviating the need to "compile" redundantly into the domain hierarchy of disorders how it should be searched. When a new domain relation is declared (e.g., a new kind of infection is added to the hierarchy) the abstract control knowledge will use it appropriately. That is, *we separate out what the domain knowledge is from how it should be used.*

Metarules were first introduced for use in expert systems by Davis (Davis, 1976), but he conceived of them as being domain-specific. In that form, principles are encoded redundantly, just like CENTAUR's control knowledge. For example, the principle of pursuing common causes before unusual causes appears as specific metarules for ordering the domain rules of each disorder (see (Clancey, 1983) for detailed discussion).

The benefits of stating metarules abstractly are illustrated further by a second example.

2.2. An implicit question-asking strategy

Another reason for ordering clauses in a system like MYCIN is to prevent unnecessary requests for data. A finding might be deduced or ruled out from other facts available to the program. For example, the rule "If the patient has undergone surgery and neurosurgery, then consider diplococcus as a cause of the meningitis" might be represented as follows.

```
PREMISE: ($AND (SAME CNTXT SURGERY)
               (SAME CNTXT NEUROSURGERY))
ACTION: (CONCLUDE CNTXT COVERFOR DIPLOCOCCUS TALLY 400)
```

We say that the surgery clause "screens" for the relevance of asking about neurosurgery. Observe that neither the relation between these two findings (that neurosurgery is a *type of* surgery) nor the strategy of considering a general finding in order to rule out one of its subtypes is explicit. An alternative way used in MYCIN for encoding this knowledge is to have a separate "screening" rule that at least makes clear that these two findings are related: "If the patient has not undergone surgery, then he has not undergone neurosurgery."

```
PREMISE: ($AND (NOTSAME CNTXT SURGERY))
ACTION: (CONCLUDE CNTXT NEUROSURGERY YES TALLY -1000)
```

Such a rule obviates the need for a "surgery" clause in every rule that mentions neurosurgery, so this design is more elegant and less prone to error. However, the question-ordering strategy and the abstract relation between the findings are still not explicit. Consequently, the program's explanation system cannot help a system maintainer understand the underlying design.

In NEOMYCIN, the above rule is represented abstractly by a metarule for the task of finding out new data.

<Domain Knowledge>

```
(SUBSUMES SURGERY NEUROSURGERY)
(SUBSUMES SURGERY CARDIACSURGERY)
```

<Abstract Control Knowledge>

```
TASK: FINDOUT
ARGUMENT: DESIRED-FINDING

METARULE002
IF the desired finding
      is a subtype of a class of findings and
      the class of findings is not present in this case,
THEN conclude that the desired finding is not present.

(IF (AND (CURRENT-ARGUMENT $SUBTYPE)
         (SUBSUMES $CLASS $SUBTYPE)
         (THNOT (SAMEP CNTXT $CLASS)))
    (NEXTACTION
        (CONCLUDE CNTXT $SUBTYPE 'YES TALLY -1000)))
```

This metarule is really an *abstract generalization* of all screening rules. Factoring out the statement of relations among findings from how those relations are to be used produces an elegant and economical representation. Besides enabling more-detailed explanation, such a design makes the system easier to construct and more robust.

Consider the multiple ways in which a single relation between findings can be used. If we are told that the patient has neurosurgery, we can use the subsumption link (or its inverse) to conclude that the patient has undergone surgery. Or if we know that the patient has not undergone any kind of surgery we know about, we can use the "closed world assumption" and conclude that the patient has not undergone surgery. These inferences are controlled by abstract metarules in NEOMYCIN.

The knowledge base is easier to construct because the expert needn't specify every situation in which a given fact or relation should be used. New facts and relations can be added in a simple way; the abstract metarules explicitly state how the relations will be used. The same generality makes the knowledge base more robust. The system is capable of making use of facts and relations for different purposes, perhaps in combinations that would be difficult to anticipate or enumerate.

3. Design criteria for NEOMYCIN

In designing an architecture for an intelligent system, we generally start with a set of behaviors that we wish the system to exhibit. For teaching, there are three dominating behavioral criteria:

1. *Solve problems.* The system should be able to solve the problems that it will teach students how to solve. This is the primary advantage of using the knowledge-based approach.[1]

2. *Explain own behavior.* The system must be able to state what it is doing: What domain knowledge it uses to solve the problem, its goals, and the methods it tries. The system might also need to state the rationale for domain facts and procedures, that is, to say why they are correct with respect to some set of constraints or assumptions (Clancey, 1983).

3. *Model student behavior.* The system must be able to recognize when its procedures and knowledge are used by another problem solver. Specifically:

   a. *The program should be able to solve problems in multiple ways.* For example, a diagnostic program should be able to cope with an arbitrary ordering of student requests for data, and evaluate arbitrary partial solutions at any time (Brown, et al., 1982a).

   b. *The program should solve problems in a manner that students can understand and emulate.* This is obviously necessary if the program is to convey a useful problem solving approach and to recognize it or its absence in other problem solvers.[2] Consequently, the problem solving procedure should be at least an ideal model of what people do, and may need to incorporate alternative and

[1] In contrast, traditional CAI programs, except in mathematics and fact recall problems such as geography, are not designed to solve problems independently. They present material, evaluate, and branch according to possible situations pre-enumerated by the program "author" (Clancey, 1982b).

[2] Of course, the knowledge-based tutor is not the only way to use computers for teaching, and the cost/benefit of this approach is not known. See (Papert, 1980, Clancey, 1982b, Brown, 1983) for discussion

non-optimal methods.[3]

Both explanation and student modeling are made possible by representing the problem solving procedure in a general way, separate from the domain knowledge, in a notation that can be translated to natural language. The ability to apply domain knowledge in an arbitrary order enables the modeling program to follow what the student is doing, as well as to make prescribe good behaviors in teaching interactions.

With emphasis on problem solving performance and only superficial explanation requirements, most early knowledge systems do not state control knowledge separately and explicitly in the manner of NEOMYCIN. Figure 3-1 summarizes the simple rule and frame-based approaches to knowledge representation. Rule-based systems have a simple (opaque) interpreter, and can easily index what facts and rules were used to make Sssertions. While the literature (e.g., (Davis, et al., 1977)) makes a major issue of the separation of the knowledge base from the "inference engine," the control knowledge is in fact implicit in the rules. Frame-based systems represent domain facts in a well-structured way, but typically control knowledge is represented in arbitrary Lisp code that cannot be explained automatically. In NEOMYCIN, meeting the behavioral goals of explanation and modeling requires a combination of approaches:

- Following the design principle of the rule-based approach, *all knowledge is stated in a simple syntax* that programs can interpret for multiple purposes. In particular, the conditional-action form of control knowledge (or any procedure) makes rules a suitable representation.

- Following the design principle of the frame-based approach, *domain knowledge is stated separately from control knowledge*, so that domain concepts and relations are explicit. In particular, the concept-relation form of domain knowledge makes frame-like structures a suitable representation. Moreover, rules are themselves organized hierarchically with additional knowledge about how they are to be applied.

Here we summarize in more detail how this architecture and the design criteria are realized in NEOMYCIN:

- The diagnostic procedure mentions no specifically medical concepts or relations.

- Clauses of domain rules are arbitrarily ordered. Most rules (136/169) have only one clause; the remaining are definitions or patterns that trigger diagnoses.

- There is no uncontrolled backchaining at the domain level. When new problem information is required to evaluate a domain rule, the program does not arbitrarily

[3] (Johnson, 1983) describes alternative models of reasoning as the basis of designing a knowledge system: mathematical optimization, simulation, and ad hoc rationalization.

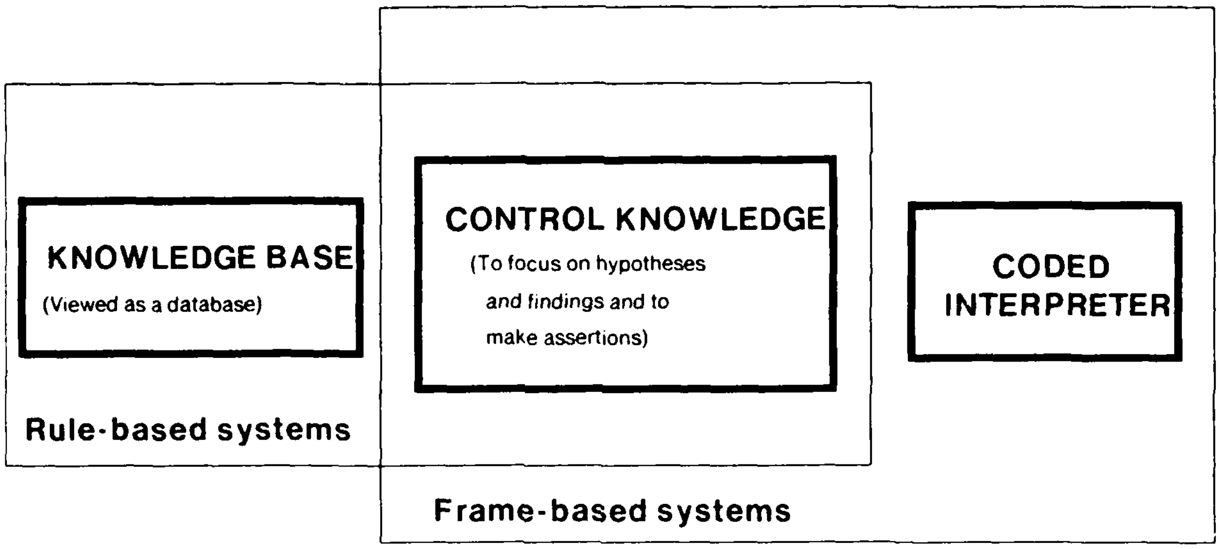


Figure 3-1: Alternative representations of control knowledge

apply domain rules, but selects among them. This is also called *deliberate subgoaling* (Laird, 1983).

- The diagnostic procedure is decomposed into subprocedures that facilitate explanation. Our rudimentary theory characterizes diagnosis as a process of shifting focus to manipulate a set of possible solutions (the "differential"), including means to generate this set, categorize and refine it, and test it for completeness (Clancey, 1984b).

- There is a fixed set of procedural (opaque) primitives:
    - ask for problem data,
    - assert a domain fact (in metarules that are generalized domain rules),
    - attempt to apply a domain heuristic rule, and
    - invoke a subprocedure.

- Domain knowledge reflects expert experience, consisting of *procedurally-defined associations*, collectively called *domain schema knowledge*:
    - *trigger rules* describe patterns of findings that immediately suggest a hypothesis, perhaps causing an intermediate question to be asked
    - *follow-up questions* are process characterizations that are immediately requested, e.g., upon finding out that the patient has a fever, the program/expert asks for the patient's actual temperature.

- o *heuristic finding/hypothesis rules* make direct connections between findings and hypotheses, omitting causal details.

- o *general questions* are broad characterizations of the patient's history that are intended to cover everything that the patient might have experienced that could cause a disease, e.g., travel, hospitalizations, medications, immunosuppression.

- The diagnostic procedure of NEOMYCIN is based on protocol analysis and previously formalized studies of medical problem solving (Clancey, 1984a, Clancey, 1984b). The procedure reflects a variety of cognitive, social, mathematical, and case experience constraints, none of which are explicit in the program.

- Our methodology assumes that there will be an accumulation of procedural knowledge over time that will be applicable in other domains.

To summarize, NEOMYCIN expands upon MYCIN's knowledge, representing domain and control knowledge separately: The domain knowledge is experiential (schemas) and the control knowledge is a general heuristic classification inference procedure.

4. Architecture of HERACLES

To make control knowledge explicit, many changes and additions were made to MYCIN. Because none of these are specific to NEOMYCIN, but are characteristics of the general framework, called HERACLES, we will hereafter refer to HERACLES and use specific examples from the NEOMYCIN knowledge base. The architecture (refer to Figure 4-3) consists of:

- control knowledge:
  - o metarules,
  - o tasks,
  - o task interpreter.
- domain knowledge:
  - o terms and relations
  - o procedural attachment (implementation specification)
  - o implementation data structures.

Briefly, a task is a procedure, namely a controlled sequence of conditional actions. Each conditional action is called a *metarule*. Metarule premises are stated in the relational language, which is indexed to domain knowledge data structures via procedural attachments. Associated with each task is additional knowledge specifying how its metarules are to be applied. The relational language we have chosen is called MRS (Genesereth, 1983, MRSDICT, 1982). Its relevant features are: a prefix predicate calculus notation, use of pattern variables with backtracking, use of backchaining in application of rules to determine the truth of propositions, and procedural attachment to allow arbitrary means for assertion or truth evaluation, thus enabling multiple representation of knowledge.

4.1. Metarules

Figure 4-1 shows a typical metarule, for the task "test hypothesis." Such rules were originally called *metarules* to distinguish them from the domain rules, and because most of them directly or indirectly have the effect of selecting domain rules to be applied. Given the set of primitive actions, the term "inference procedure rule" is more accurate.

Metarule premises consist of a conjunction of propositions. There are three kinds of relations: domain, problem solving history, and computational functions. In addition, a relation may be a composite inferred from rules, which we call a *metarule premise relation* (see Figure 4-2). The indicated metarule will collect the set of unapplied domain rules that mention an unrequested finding that is a necessary cause to the hypothesis under consideration (e.g., receiving antibiotics is a necessary cause of partially treated meningitis).

```
Premise: (MAKESET (ENABLING.QUESTIONS CURFOCUS $RULE) RULELST)

Action:  (TASK APPLYRULES RULELST)

Task:    TEST-HYPOTHESIS
```

Figure 4-1: Typical NEOMYCIN metarule.

```
Premise:  (AND (ENABLINGQ $HYP $FOCUSQ)
               (NOT (TRACEDP ROOTNODE $FOCUSQ))
               (EVIDENCEFOR? $FOCUSQ $HYP $RULE $CF)
               (UNAPPLIED? $RULE))

Action:  (ENABLING.QUESTIONS $HYP $RULE)
```

Figure 4-2: Typical rule for concluding about a metarule premise relation

4.1.1. Groundwork: Domain knowledge, problem-solving history, and Lisp functions

In creating HERACLES from EMYCIN, the original "rule," "context," and "parameter" structures were given new properties, but remain as the primitives of the knowledge representation. For example, parameters are now called *findings* and *hypotheses*, depending on whether they are supplied to the program or inferred. They are hierarchically related. Rules are annotated to indicate which are definitions and the direction of causality if appropriate. Inverse pointers are automatically added throughout to allow flexible and efficient indexing during problem solving. (See Appendix I for a complete listing of domain terms and relations. Further discussion appears in (Clancey, 1984a) and (Clancey, 1984b).)

The problem solving history consists of the bookkeeping of EMYCIN (record of rule application and parameter determination), plus additional information about task and metarule application.

Simple Lisp functions, such as arithmetic functions, are not translated into tasks and metarules. In addition, complex functions in the modified domain (EMYCIN) interpreter, such as the rule-previewing mechanism, are still represented in Lisp. Metarules also invoke a few complicated interface routines, for example to allow the user to enter a table of data.[4]

The domain data structures, problem-solving history, and Lisp functions are collectively called the *Heracles groundwork*. Procedural attachment is used to interface relations appearing in metarules with the HERACLES groundwork.

4.1.2. Procedural attachment--Motivation and advantages

A key improvement of HERACLES over early versions of NEOMYCIN is the use of a relational language with procedural attachment in the metarules. Formerly, arbitrary Lisp functions were invoked by metarule premises, inhibiting explanation and student modeling because this code could not be interpreted easily by programs (excluding, of course, the implementation-level Lisp interpreter and compiler).

The advantages of using MRS for representing the premises of metarules are:

- Tasks and metarules themselves make procedural steps and iteration explicit; the MRS encoding makes metarule premises explicit.

- Prefix Predicate Calculus (PPC) provides a simple syntactic structure of relations, terms, and logic combinations that facilitates writing programs to interpret it.

- Backtracking with variables allows matching constraints to be stated separately from the operations of search and database lookup, so complex interactions are restated as simple conjuncts.

- Using rules to infer metarule premise relations allows intermediate levels of abstraction to be cogently represented as rules.

- The messy implementation details of the domain and problem solving history data structures are hidden, yet these underlying structures make efficient use of storage and are indexed for efficient access.

- The PPC syntax facilitates stating metaknowledge. Patterns among relations can be stated explicitly, making it possible to write interpretation procedures that treat relations abstractly. For example, the metarule compiler need only reason about the

---

[4]The original EMYCIN function FINDOUT *was* made into a task to enable the explanation and modeling programs to account for how HERACLES infers findings when it doesn't ask the user. In addition, portions of the rule interpreter and all forward reasoning (e.g., application of domain antecedent rules) are re-represented as metarules.

half-dozen categories of domain data structures, rather than deal with the relations directly. This idea is central to the possibility and advantage of using abstract procedures (see Section 9).

At run time an MRS program deduces what method (inference procedure) to use to evaluate the truth of a statement, and then applies the indicated function. The original program we implemented in MRS, called MRS/NEOMYCIN, was much too slow to be practical (see Section 6). In the current version of HERACLES the metarules are compiled, replacing all of the relations and procedural attachments with direct lookup in the domain knowledge base and problem-solving history (Section 5). To do this, the compiler needs information about how each relation is actually represented in the Lisp data structures and functions of the HERACLES groundwork. Table 4-1 gives the possible forms of implementation, with examples.

implementation	*example*	*interpretation*
FLAG	NEW-DIFFERENTIAL	T or NIL Lisp var
VARIABLE	STRONGCOMP.WGHT	Lisp variable
LIST	DIFFERENTIAL	Lisp list
PROPMARK	ASKFIRST	T or NIL property
PROPLIST	CHILDREN	list-valued property
PROPVAL	PROMPT	arbitrary property
METARULE-PREMISE-RELATION	TAXREFINE?	determined by a rule
FUNCTION	SAMEP	Lisp function

Table 4-1: Implementation of HERACLES groundwork

There is a miscellaneous category of relations for which the compiler produces relation-specific code. Examples are: a relation for satisfying another relation as many times as possible (MAKESET); the quadruple relation among a finding, hypothesis, domain rule, and certainty factor (EVIDENCEFOR); and relations that can be easily optimized by the compiler (e.g., MEMBER and NULL).

In summary, we use a relational specification for perspicuity, Lisp data structures and functions for convenience and efficiency, and compile the metarules to avoid the expense of run-time pattern matching and indirect lookup. The resulting metarules are easier to read, maintain, and explain automatically than the original Lisp code.

4.2. Tasks

A HERACLES *task* consists of an ordered sequence of metarules and additional knowledge about how they should be applied, used by the *task interpreter*. The program begins by executing the top-level task, to carry on a consultation. The tasks then direct the application of metarules; when metarules succeed, primitive actions are taken and other tasks invoked. See Figure 4-3.

Currently there are 75 metarules in HERACLES, organized into 40 tasks. The invocation structure, integrating hypothesis- and data-directed reasoning, is shown in Figures 4-4 and 4-5.

```
                         |-------------------------|   Task invocation
                         |    Task interpreter     |<--------------------+
                         |                         |------------------+  |
                         |-------------------------|                  |  |
                                   | ↑                                 |  |
                                   V |                                 |  |
                     |--------------------------------|                |  |
                     |            Metarules           |                |  |
                     |     (IF <premise>   <action>)  |                |  |
                     |---------- | ↑ -----  | ↑ -------|                |  |
                                 | |        | |                        |  |
                                 V |        V |                        |  |
          |-------------------------|  |-------------------------|    |  |
          |        Metarule         |  |        Metarule         |    |  |
          |   premise functions     |  |    action functions     |    |  |
          |-------------------------|  |-------------------------|    |  |
              | ↑                         | ↑       | ↑       | ↑      |  |
              V |                         V |       V |       | |      |  |
Examine: Domain rules & relations     Domain rule  Data requests  | +----+  |
         Problem-solving history:     application  & assertions   +--------+
           domain and task
```

Figure 4-3: Flow of control in HERACLES

4.3. The task interpreter

The information used by the task interpreter is:

- The *task focus*, which is the argument of the task (e.g., the focus of the task TEST-HYPOTHESIS is the hypothesis to be tested). Only one focus is allowed.

- The main body of ordered metarules, which are to be applied to complete the task. (Called the *do-during* metarules.)

- The *end condition*, which may abort the task *or any subtask* when it becomes true. Aborting can occur only while the do-during metarules are being applied. The end condition is tested after each metarule of a task succeeds. A task may also be marked to prevent abortion.

- Ordered metarules to be applied before the do-during rules.

- Ordered metarules to be applied after the do-during rules.

- The *task type*, which specifies how the do-during metarules are to be applied. There are two dimensions to the task type: *simple* or *iterative*, and *try-all* or not-try-all. The combinations give four ways of applying the do-during rules:

    - *Simple, try-all.* The rules are applied once each, in order. Each time a metarule succeeds, the end condition is tested.

    - *Simple, not-try-all.* The rules are applied in sequence until one succeeds or

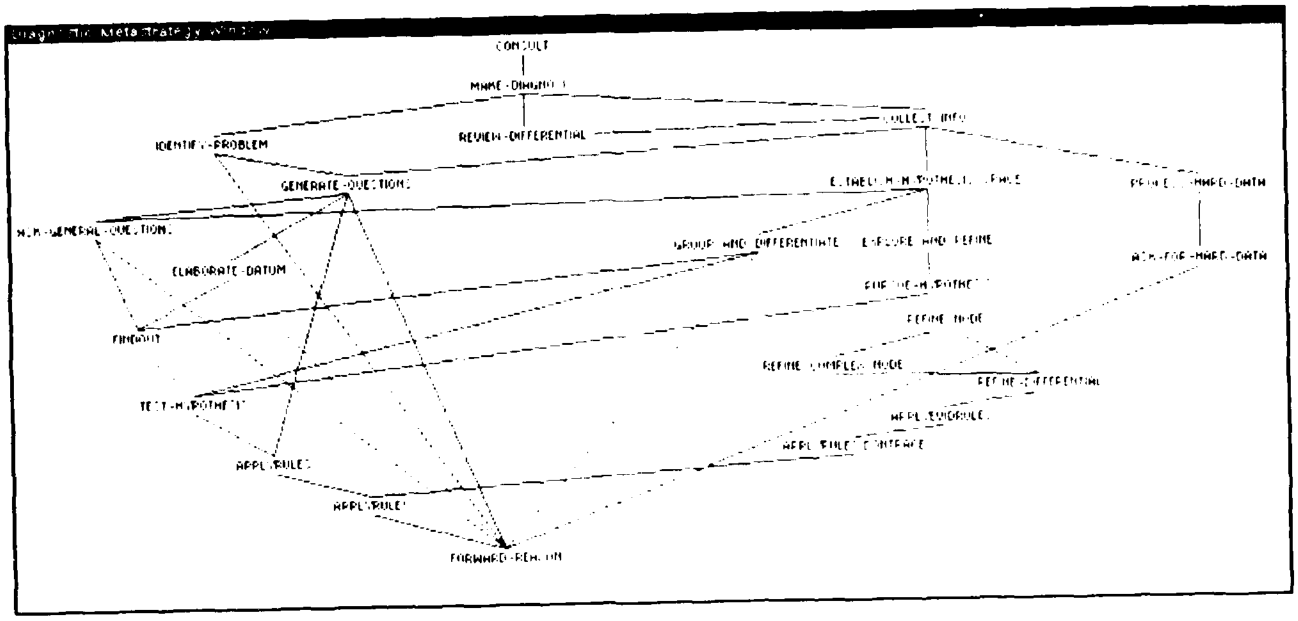


Figure 4-4: Heracles classification tasks
[shown as a lattice]

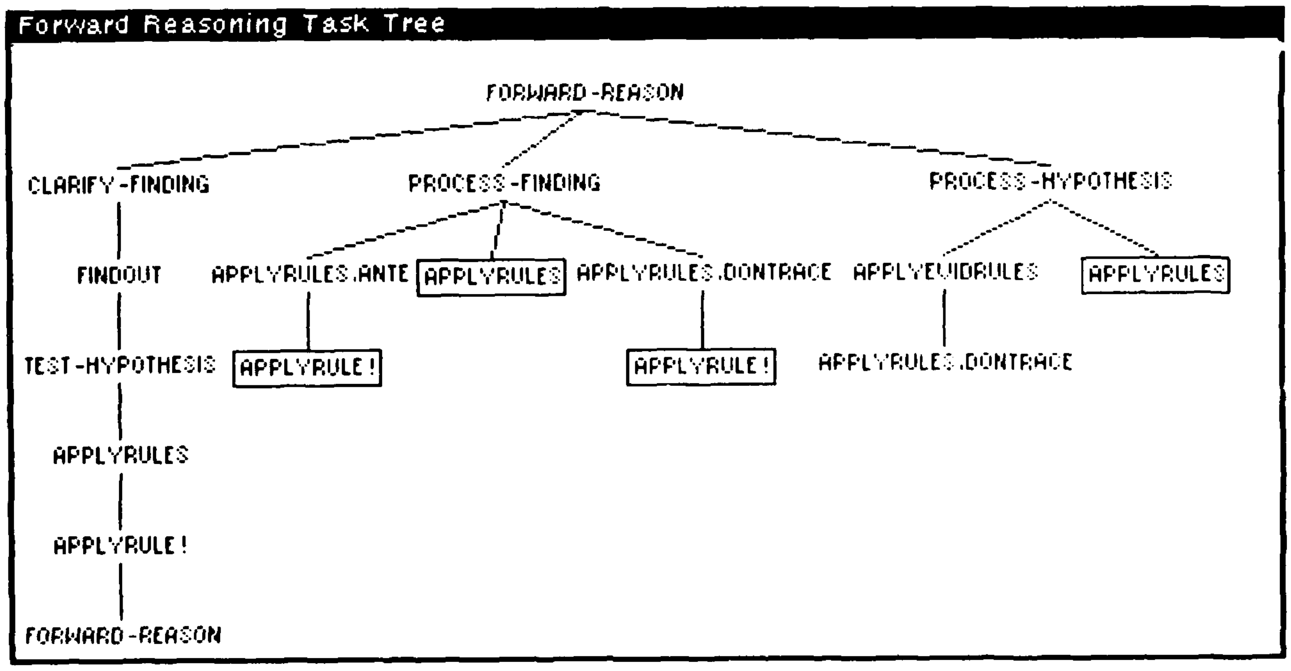


Figure 4-5: Heracles forward reasoning tasks
[shown as a hierarchy]

the end condition succeeds.

- *Iterative, try-all.* All the rules are applied in sequence. If there are one or more successes, the process is started over. The process stops when all the rules in the sequence fail or the end condition succeeds.

- *Iterative, not-try-all.* Same as for iterative try-all, except that the process is restarted after a single metarule succeeds.

The four combinations of control are shown in Figure 4-6 with their common equivalents.

	not try-all	try-all
Simple	COND	PROG
Iterative	Pure Production System	"For loop"

Figure 4-6: Common equivalents for four ways of controlling metarules

The experiment of representing the task interpreter in *metacontrol* rules is described in Section 6.

Only a few tasks have end conditions. Studying them reveals the following interpretations:

1. *The end condition is the negation of a pre-requisite for doing the task.* For example, the pre-requisite of examining subcategories of hypotheses (EXPLORE-AND-REFINE) is that all more general categories have already been considered. The end condition of EXPLORE-AND-REFINE is (NOT (WIDER-DIFFERENTIAL)), indicating that there is no new hypothesis that lies outside of previously considered categories.

2. *The end condition is the goal the task seeks to accomplish.* For example, the goal of probing for additional information (GENERATE-QUESTIONS) is to suggest new hypotheses. The end condition of GENERATE-QUESTIONS is (DIFFERENTIAL $HYP), indicating that the program has at least one hypothesis under consideration.

Figure 4-7 gives the metarules and control information for the task GENERATE-QUESTIONS (with auxiliary rules to conclude about metarule premise relations). Figure 4-8 shows a partial history of task invocation for a typical NEOMYCIN consultation (After gathering laboratory data (PROCESS-HARD-DATA, PHD), new hypotheses are shown being explored. PURSUE-HYPOTHESIS (PUH) Is invoked three times, leading to application of four domain rules and an attempt to find out five findings.)

```
GENERATE-QUESTIONS

TASK-TYPE:        ITERATIVE
ENDCONDITION:     ADEQUATE-DIFFERENTIAL
LOCALVARS:        ($FOCUSPARM RULELST)
ACHIEVED-BY:      (RULE003 RULE359 RULE386 RULE425)
ABBREV:           GQ

RULE003

Premise:  (NOT (TASK-COMPLETED ASK-GENERAL-QUESTIONS))
Action:   (TASK ASK-GENERAL-QUESTIONS)
Comment:  Ask general questions if not done already.

RULE359

Premise:  (PARTPROC.NOTELABORATED? $FOCUSPARM)
Action:   (TASK ELABORATE-DATUM $FOCUSPARM)
Comment:  Ask for elaborations on partially processed data.

RULE386

Premise:  (MAKESET (PARTPROC.SUGGESTRULES? $RULE)
                   RULELST)
Action:   (TASK APPLYRULES RULELST)
Comment:  Apply rules using known data as if they were trigger rules.

RULE425

Premise:  (NOT MORE-DATA-COLLECTED)
Action:   (DO-ALL (COLLECT.MORE.DATA)
                  (TASK FORWARD-REASON))
Comment:  Simply ask the user for more information.

Auxiliary metarule premise rules

Premise:  (AND (PARTPROC.DATA $DATUM)
               (YNPARM $DATUM)
               (SAMEP ROOTNODE $DATUM $CF)
               (NOT (ELABORATED $DATUM))
               (OR (PROCESSQ $DATUM $ANY)
                   (SUBSUMES $DATUM $ANY)))
Action:   (PARTPROC.NOTELABORATED? $DATUM)

Premise:  (AND (PARTPROC.DATA $PARM)
               (SUGGESTS $PARM $SUGHYP)
               (EVIDENCEFOR? $PARM $SUGHYP $RULE $CF))
Action:   (PARTPROC.SUGGESTRULES? $RULE)
```

Figure 4-7: Metarules and control information for task GENERATE-QUESTIONS

5. The metarule compiler: MRS -> Interlisp

Originally in MRS/NEOMYCIN, the MRS interpreter was used for pattern matching, including resolution, backtracking, and procedural attachment. In addition, the task interpreter itself was encoded in MRS rules, controlled by a simple deliberation-action loop (described in Section 6). Even after we reverted to a Lisp task interpreter for efficiency, with the metarules accessed directly as Lisp structures, the program was still too slow to use. Finally, after studying the

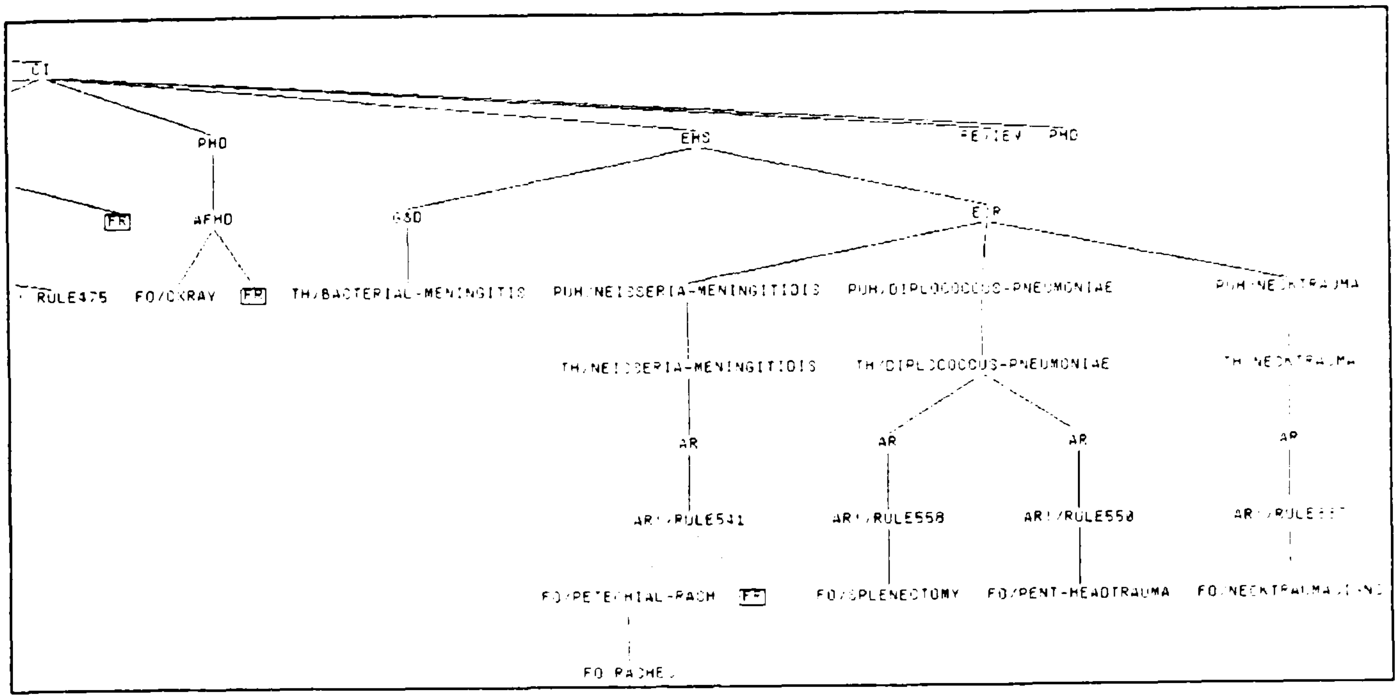


Figure 4-8: Excerpt of task invocation for a typical NEOMYCIN consultation
--tasks are abbreviated; task foci appear as medical terms

rules and hand-written Lisp code equivalents, we found that it was possible to compile the metarule premises, including metarule-premise relations and their rules and procedural attachments into ordinary Lisp code.

In the compiled version, each metarule premise generally becomes a Lisp function; each relation concluded by a metarule premise relation rule (Figure 4-2) becomes a Lisp function. In general, it is difficult to write a compiler for MRS-style rules. However, several features of our rules and a few simplifications made it easy to write the compiler:

- Only one rule concludes about each metarule premise relation. Where necessary, rules were combined into a single rule with a disjunction.

- Relations are either predicates or functions, rather than being used in both ways. For example, (CHILDREN $HYP $CHILD) is only used as a function [illegible] never as a predicate to test whether a given candidate is a child [illegible] hypothesis. This was not a deliberate design choice--all of the 166 relations in HERACLES satisfy this property.

- Functional relations are all single-valued (except EVIDENCEFOR). Consequently, backtracking (to find matches for variables in conjunctions) can be expressed as nested *find* or *therexists* loops; failure of the inner loop and return to the next outer loop for a new variable match is equivalent to backtracking.

- Rule conjuncts are ordered manually so that a variable is found (by a functional relation) before it is tested (by a predicate relation). This is a natural way to write the rules.

- Inverse relations are chosen so that Lisp atom with the property corresponding to the relation is the first variable in the relation. For example, the functional relation CHILDREN, as in (CHILDREN $HYP $CHILD), is used when $HYP is known. Again, this occurred naturally rather than being a deliberate design choice.

- As a trivial simplification, redundant clauses are not factored out of disjuncts, in the form
  ```
  (AND <common clauses> (OR d1 d2 ... dn)).
  ```
  Consequently, a few rules are slightly awkward.

5.1. An example

To compile the metarule shown in Figure 4-1, the compiler checks to see what kind of relation ENABLING-QUESTIONS is. Here is what it discovers:

```
ENABLINC.QUESTIONS

IMPLEMENTATION: METARULE-PREMISE-RELATION
MULTIPLEMATCH:  T
USED-BY:        (RULE566)
UPDATED-BY:     (RULE9325)
```

In writing a function to replace rule9325 (Figure 4-2), the compiler observes that this is a functional relation (not a PREDICATE), so it must return $RULE. Moreover, it is MULTPLEMATCH, so all matches for $RULE are returned. The code appears in Figure 5-1. The main difficulty is keeping track of what variables are bound and knowing when to do an iteration versus simply checking if a match exists. For example, if $RULE were an argument to this function, the code would have been very different.

```
(ENABLING.QUESTIONS
  [LAMBDA ($HYP)
    (PROG ($RULE)
          (RETURN (for $FOCUSQ
                       in (GETP $HYP (QUOTE ENABLINGQ))
                       join (AND (NOT (TRACEDP ROOTNODE $FOCUSQ))
                                 (for RULECFLST
                                    in (EVIDENCEFOR $FOCUSQ $HYP)
                                    as $RULE is (CAR RULECFLST)
                                    collect $RULE
                                    when (UNAPPLIED? $RULE])
```

Figure 5-1: Compiler-generated code for metarule premise rule shown in Figure 4-2

In general, the compiler's code is a little easier to understand than the manually-written original because it doesn't use constructs like *thereis* and *never*, which require some mental gymnastics to logically invert and combine.

5.2. More details

The compiler recursively pieces together code for each relation. A second pass ensures that a value is returned from inner loops and Lisp variables are properly bound. Important subprocedures:

- Produce code for the EVIDENCEFOR relation, (EVIDENCEFOR $FINDING $HYPOTHESIS $RULE $CF)--$HYPOTHESIS and $RULE or $RULE alone might be unknown at the time of matching.

- Gather and compile clauses that test a particular variable (the iterative variable for a compiled loop).

- Modify "find" iterations (on the compiler's second pass) to return the correct value, changing *suchthat* to *collect* or *join*, etc.

Besides the IMPLEMENTATION property described in Section 4.1.2, relations may have PREDICATE and MULTIPLEMATCH properties. PREDICATE only applies to relations that have an implementation of METARULE-PREMISE-RELATION or FUNCTION (ordinary LISP function). MULTIPLEMATCH only applies to a metarule-premise-relation. It means that the MRS rule should be "matched as many times as possible." In essence, the compiler changes *find <var> in <list> suchthat...* to *for <var> in <list> collect <var> when....*

Different code is produced depending on whether the result variable, the last variable in the proposition, is bound when a clause is compiled. For example, the code for the clause (TRIGGERPARMS $RULE $PARM) might be:

- (find $PARM in (GETP $RULE 'TRIGGERPARMS) suchthat ...) if $PARM is not bound yet,

- (FMEMB $PARM (GETP $RULE 'TRIGGERPARMS)), if $PARM is bound, or

- (GETPROP $RULE 'TRIGGERPARMS) if $PARM is not bound and it is not tested later in the rule.

All relations compile in a similar way--finding a variable, testing it, setting it, or simply checking to see if a value exists.

Looking for tests to place in the "suchthat" part of an iteration is tricky. The compiler obviously must include later clauses that mention the iteration variable. But it must also include earlier clauses that set a variable mentioned in these later tests, plus later tests of such variables.

The relation MAKESET is used in many metarules. It expects its inner relation to be of type MULTIPLEMATCH, returning a list. Many rules also use a variation of SETQ that only sets the result variable if it is non-NIL. This is convenient because usually in a metarule premise this variable is the focus of the task. We don't want to lose the old focus until we get a new one.[5]

In conclusion, we have found that the PPC notation as a specification language for metarules is convenient, intuitively natural, and allows efficient compilation. The development of the explanation program (Section 7) and the subsequent study of patterns in the domain relations (Section 9) reveal that the notation also has unexpected advantages for helping us to understand the nature of procedures.

6. MRS/NEOMYCIN: An experiment in explicit metacontrol

In a program called MRS/NEOMYCIN we attempted to use MRS in a direct way to represent the task interpreter in rules.[6] Our intention was that the new representation would lead to better explanation capability, as well as enhance the debugging capability for knowledge acquisition. This experiment failed because the linear sequence of rules into which we translated the interpreter disguises the iterative control so that it is difficult to read, maintain, and explain automatically. In this section we briefly summarize the implementation and reflect on what we learned about making procedures explicit.

6.1. MRS/NEOMYCIN implementation

To specify procedural rule application in rules, MRS was augmented with a form of backward chaining (called "RULEFOR") that dynamically calculates rules to be used in deduction. Specifically, *metacontrol rules* calculate what metarules to use at any time. At the top is a deliberation-action loop. In principle, deliberation only involves domain knowledge and problem-solving state lookup; assertions about the problem are only made by metarule and domain rule actions. Specifically, we forbid hidden side-effects, such as saving computations in rule premises for use by rule actions. Adhering to this discipline simplifies the explanation program and other uses of the control knowledge. Modifications were made to MRS to cope with recomputation inefficiencies that resulted (allowing caching and a history of rule application).

Some of the elements of MRS/NEOMYCIN have survived in HERACLES and have already been described: the relational specification of domain knowledge and problem solving history; metarule premise relations; and procedural attachment (which survived as IMPLEMENTATION categories, rather than the original procedures for asserting, unasserting, and evaluating the

[5]In MRS/NEOMYCIN a stack was maintained by the deliberation-action loop, obviating the need for Lisp variables and allowing the focus to be retrieved and reset in a cleaner way (Section 6).

[6]The design and implementation of MRS/NEOMYCIN is primarily the work of Conrad Bock, in partial fulfillment of the Master's degree in Artificial Intelligence at Stanford University.

truth of statements). The additional constructs in MRS/NEOMYCIN that we used for representing metacontrol in predicate calculus are:

- *Metacontrol rules* (MC "metacontrol" and DR "do-during rulefor" rules), corresponding to the primitive conditional actions of the task interpreter--apply metarules for a task, detect the end condition, and do bookkeeping.

- *Stack of tasks and focus arguments* so that the task interpreter can be invoked recursively (tasks can invoke other tasks).

- *History* of metarules and metacontrol rules applied or failed in a task (for bringing about sequential and iterative computation).

- *Metametacontrol rules* (the NR "nextaction rulefor" rules) that refer to the metacontrol rules by name and invoke them in the proper sequence (do-before, do-during, do-after, then bookkeeping), allowing a normal backchaining interpreter to be used at the highest level.

- *Deliberation-Action loop*, a small Lisp program that invokes MRS to deduce what metarule action to do next.

As can be seen, this is a fairly complex framework for reasoning about control, greatly elaborating upon Davis's original idea of using metarules for refining search (Davis, 1980). In particular, domain-independent metarules invoke base-level rules, MC and DR rules order and choose metarules, and NR rules control the ordering process. This framework, a re-representation of the NEOMYCIN task interpreter, is especially of interest because it provides a control language within MRS in terms of a *tasking mechanism.* We believed that this architecture and its primitives would significantly increase the usefulness of MRS for representing control knowledge.

6.2. Problems with perspicuity and efficiency

In constructing MRS/NEOMYCIN, we invented a way to shoe-horn a complex procedure, the task interpreter, into the deductive mechanisms of MRS, using MRS's metalevel control mechanism (deducing how some thing should be deduced) to control application of metarules and access to domain and problem solving history structures. However, our design was impractical: It was an order of magnitude two slow for available (1982) computers, and most serious of all, the resulting program is conceptually more difficult to understand. What went wrong?

The chief deficiency of MRS/NEOMYCIN is the complexity of the rules representing the task interpreter. It is difficult for people to understand nested iteration that is expressed as levels of rules controlling other rules. Certainly, a program would be in no better position to understand the task interpreter from this kind of specification.

The representation used in MRS/NEOMYCIN was developed to allow machine interpretation, specifically, to allow the augmented MRS reasoning mechanisms to deduce what metarules should be applied at what time. Similar to the goals of AMORD (de Kleer, 1979), we wanted a representation that would make control knowledge explicit in the form of assertions about the control state. While this was achieved, we must not confuse MRS/NEOMYCIN's ability *to carry out the procedure* with understanding it. Predicate calculus is often proposed as a means of making knowledge explicit, but just *stating the steps of the procedure in rules* doesn't make the *meaning of the process* explicit. For example, referring to Figure 6-1, consider the difficulty of understanding that NR rule 4 and DR rule 3 bring about simple try-all application of metarules.[7] The abstract properties of the program's *output* are not immediately obvious from the primitive terms and their combination in rules. AMORD's rules suffer from the same problem: levels of abstraction are missing that would make it clear what the rules and control primitives are doing. This level of specification is analogous to an assembly level program.

NEXTACTION RULEFOR rule 4 (NR-4)

```
    [IF (AND (CURRENT-TASK $CURTASK)
             (TASKTYPE $CURTASK SIMPLE)
             (TASK-TRY-ALL $CURTASK)
             (THNOT (APPLIED-IN-TASK $CURTASK MC-4)
        (RULEFOR (NEXTACTION $ACTION) MC-4]
```

MetaControl rule 4 (MR-4)

```
    (IF (AND (DODURING $ACTION)
             (DONT-STOP-TASK $CURTASK))
        (NEXTACTION $ACTION))
```

DODURING RULEFOR rule 3 (DR-3)

```
    (IF (AND (CURRENT-TASK $CURTASK)
             (TASKTYPE $CURTASK SIMPLE)
             (TASKRULE $CURTASK $MRULE)
             (THNOT (APPLIED-IN-TASK $CURTASK $MRULE)))
        (RULEFOR (DODURING $ACTION) $MRULE))
```

Figure 6-1: Control rules specifying "simple, try-all" metarule application

We observe three different levels for understanding a procedure:

- *Understanding what the procedure is:* an ability to execute (or simulate) the procedure, that is, to compute the result of applying the steps of the procedure over time;

- *Understanding what the procedure accomplishes:* an ability to describe patterns in the result *abstractly* in terms of the procedure's overall design or goal, and

[7]To infer a NEXTACTION, NR 4 causes Metacontrol rule 4 to be applied once for simple, try-all tasks. To infer a DODURING action, concluded by a metarule, DR 3 then selects metarules that haven't been applied. The combined effect is that each metarule is applied once.

- *Understanding why the procedure is valid:* an ability to relate the design of the procedure to its purpose and constraints that affect its operation, that is, understanding the rationale for the design.

The same distinctions apply to the metarule level, and are perhaps more easily understood there. For example, consider the actions of HERACLE's metarules for focusing on hypotheses. First it pursues siblings of the current focus, then it pursues immediate descendents. This is transparent from the ordering of the metarules and the relations mentioned in the premises. Thus, the language adequately expresses the *execution knowledge* of the desired procedure. Now, if you simulate this two step procedure in your mind, you will see a pattern that we call "breadth-first search." MRS/NEOMYCIN can certainly "read" these steps, but it doesn't assign the concept "breadth-first search" to this pattern. Knowing the abstract definition of "breadth-first search," you are able to verify that the program has this design by identifying and classifying patterns in what the program does. The definition of "breadth-first search" and its relation to these two metarules, the *design knowledge* of the procedure, is not expressible in MRS/NEOMYCIN. Finally, mathematical properties of hierarchies and the goal of making a correct classification efficiently, constrain the program, suggesting this design choice. This *rationale knowledge* is also not expressible in MRS/NEOMYCIN. Thus, knowing why the program does what it does requires two kinds of inference: abstraction to characterize patterns and a proof arguing that this design satisfies certain constraints.

The original LISP code of the task interpreter, with abstract concepts such as "repeat-until," "first," and "finally," is more readable to a programmer than the metacontrol rules. All we have gained is a simple interpreter that disciplines design of the system, the deliberation-action loop. Our other goal, writing a program with the flexibility that enables reasoning about control (allowing for dynamic changes in metacontrol), could hardly be said to have been accomplished. How could MRS/NEOMYCIN automatically integrate incremental additions to its control knowledge without an understanding of the procedure's design and rationale? This same criticism can be made about the statement of control regimes given by de Kleer in AMORD and by Genesereth in MRS (Genesereth, 1983). MRS/NEOMYCIN and these other programs are in no better position to explain what they are doing than programs written in Lisp. These programs can make fine-grained statements about what they are doing, but they have no conception of the abstract design that lies behind their actions nor the motivation for this design.

A solution to the problem of readability (for understanding design) is use a higher-level language and a compiler, as in traditional programming languages. The task language is itself a step in the right direction, but tasks merely *name* subprocedures. Reasoning about them and changing them requires additional knowledge about what the tasks mean and how the metarules accomplish them. We take up this issue further in Section 9. Moreover, even if we wrote the task interpreter in the task language itself, and wrote a compiler that converted this language directly into MRS control rules, we would need yet another compiler to make the MRS implementation fast enough to be practical.[8]

In summary, we achieved a "declarative" representation of the task interpreter, *with respect to the* MRS *interpreter*--all control knowledge is represented uniformly as a set of rules, facts in a propositional database, and procedural attachments. But the representation leaves out levels of abstraction that people find to be useful for understanding the design of a procedure, namely listing conditional actions in sequence, when appropriate, and indicating the control for iteration over sequences ("for loops"). These are the two main characteristics of the HERACLES task language, or rule sets in general, accounting for the value of this construct.

7. Use of abstract procedures in explanation and teaching

7.1. Procedural Explanations

Our first explanation program for HERACLES (Hasling, 1984) demonstrated how the MYCIN HOW/WHY line of reasoning capability could be adapted to describing the inference procedure. With translation of metarules premises to PPC, we are now able to generate much more detailed, selective descriptions of reasoning. Figure 7-1 demonstrates this capability.

```
 Has Mary taken medications recently?
** WHY

We are attempting to determine whether Mary has received
antimicrobial medication.

Antimicrobial medication is a necessary factor in causing
partially-treated-bacterial meningitis, a hypothesis we are
attempting to confirm.
```

Figure 7-1: Excerpt of HERACLES explanation

As opposed to simply reading back tasks and metarules in response to successive WHY questions, the new program reasons about which tasks and which metarule clauses to mention. The explanation heuristics include omitting: tasks with rules as arguments, computational relations, relations believed to be known to the user, and relations that might be inferred from known relations. The first time a term in a relation is mentioned (usually a finding or hypothesis), it is described in terms of the active task for which it is a focus, if any. Using a second pass, introduction of pronouns and more complex sentence structure is possible. This explanation should be contrasted with the MYCIN-style response, which involves merely printing the domain rule the program is currently applying, with no strategic or focusing structure. Given the abstract inference procedure, it is straightforward to describe reasoning in terms of

[8]The slowness of MRS/NEOMYCIN is chiefly due to the indirection inherent in retrieving procedural attachments and unnecessary recomputation. For example, each time that a metarule succeeds in providing an action to the deliberation-action loop, the list of metarules is re-deduced, leaving out the ones that have been applied already. The program achieves sequential application in this painful way because it is missing the metaknowledge that the list of applicable rules will never change.

previous explanations, for example, "We are still trying to determine..." or "Following the same principle as in question 5, we are...." Development of the explanation system to exploit the new representation in this manner has just begun.

7.2. Strategic modeling

We have developed a prototype student modeling program called IMAGE; it interprets HERACLES' metarules to explain a sequence of data requests using a mixture of top-down predictive simulation and bottom-up recognition (London and Clancey, 1982). The capabilities of the program are not clear because the space of possible models and heuristic modeling operators have not been precisely defined. Yet, the program does demonstrate the value of stating the diagnostic procedure in a language that can be interpreted by multiple programs. Originally, we believed that making the procedure explicit would be important for teaching it to students. However, now it is becoming clear that the main value of the architecture is for modeling missing domain knowledge or detecting misconception. By observing over time, the program will be able to detect that the student knows a procedure *in general*, so when the student's behavior diverges from the program's, it can infer how his factual knowledge is different. Thus, we make a distinction between knowing a procedure and having factual knowledge that allows applying it to a specific problem situation.

It is also becoming clear that the modeling program needs additional knowledge about the metarules. For example, it is useful to know what metarules it might make sense to delete or reorder. In some cases, leaving something out indicates a different preference for ordering choices; in other cases it indicates a different procedure entirely. This additional information would also help a tutoring program know which deviations from HERACLES' behavior are worth bringing to the student's attention.

7.3. Guidon2 programs

A family of teaching programs, collectively called GUIDON2, are under development that use NEOMYCIN as teaching material, analogous to the way GUIDON was built on MYCIN. The first of these programs is called GUIDON-WATCH; making use of sophisticated graphics for watching NEOMYCIN solve a problem (Richer and Clancey, 1985). For example, the program highlights nodes in the disease networks to show how the search strategy "looks up" to categories before it "looks down" to subtypes and causes. Other programs on the drafting board would allow a student to issue task/focus commands and watch what NEOMYCIN does, explain NEOMYCIN's behavior, and debug a faulty knowledge base. This work has all been directly inspired by Brown's proposals for similar tutoring environments for algebra (Brown, 1983).

8. Generalization: Other applications

8.1. CASTER: A knowledge system built from HERACLES

As a test bed for developing HERACLES and testing its generality, a small knowledge system has been built that diagnoses the cause of defects in cast iron (using molds made from sand)[9]. With the diagnostic procedure already in place, we needed only to define the domain knowledge, using the relational language. We defined a hierarchy of disorders in terms of stages in the process of metal casting (analogous to NEOMYCIN's etiological hierarchy of diseases) and a causal network relating findings to etiologies. It is evident now that we don't understand very well the principles for constructing such a causal network in a consistent and complete way. The HERACLES framework is helping us to focus on these and other domain relations that need to be articulated in more detail.

8.2. Using the task language for other procedures

Recognizing that the task/metarule language could be developed into a good high-level language for procedures in general, whose structure could be exploited for explanation, modeling, etc., we are writing our new explanation program in terms of tasks and metarules. The explanation program is *constructive*, because it pieces together what to say rather than selecting it whole from built in responses. Consequently, some kind of database is needed for posting partial explanations, as well as operators to examine and modify the evolving response. An example explanation rule, for the task that decides what relations in a HERACLES metarule to mention, is given in Figure 8-1.

```
Premise: (AND (RELTYPE? $REL 'COMPUTATIONAL)
              (NOT (USER-KNOWN $REL))
              (NOT (EXPLAINED-IN-SUBDIALOG $REL))
              (PREFERENCE THIS-USER 'COMPUTATIONAL-DETAILS))

Action:  (MENTION $REL TASK-INSTANCE)
```

Figure 8-1: An explanation heuristic rule

Our experience indicates that some modifications to the task language may be useful, such as allowing multiple arguments to a task and designating some tasks to be generators that take a list and pass elements on to a subtask in order (several tasks in HERACLES, for example, APPLYRULES, are of this form). Ultimately, we believe that it will be useful for the design of the explanation procedure to be explicit enough to allow it to reflect upon itself to produce alternative explanations when the student does not understand the first response.

[9]This project is primarily the work of Tim Th[illegible]an

9. Studying abstract procedures and relations

To review, in HERACLES there are approximately 75 metarules that constitute a procedure for doing diagnosis. The metarules reference:

- Domain knowledge: types of findings and hypotheses, and relations among them;
- Control knowledge about metarules and tasks:
    - (static) the argument of a task, whether metarules are to be applied iteratively, when to return to the head of the list, and the condition under which a task should be aborted,
    - (dynamic) whether a task completed successfully, whether a metarule succeeded or failed, etc.
- Domain problem-solving history: "active" hypotheses, whether a hypothesis was pursued, cumulative belief for a hypothesis, hypotheses that "explain" a finding, rules using a finding that are "in focus", a strong competitor to a given hypothesis, etc.
- Computational predicates and functions: comparison, numerical, and primitive inference routines.

These concepts form the vocabulary for a model of diagnosis, the terms in which expert behavior is interpreted and strategies are expressed (see Appendix I for a complete listing). This vocabulary of *structural relations* and the *body of abstract control knowledge* can itself be studied, as well as applied in other problem domains. It is the beginning of a "descriptive base" (cf. (Miller, 1983), page 182) from which generalizations can be made about the nature of a useful knowledge organization for problem solving.

In the sections that follow we consider a number of issues about the nature of relations and abstract procedures, emphasizing how relations or classifications define procedures and how these relations are derived.

9.1. The nature of an abstract inference procedure

The idea of an abstract procedure is directly related to the idea of separating programs from data, which itself derives from idea of general mathematical laws and formulae. An abstract procedure is one in which problem-specific values are replaced by terms characterizing the type of each value, which become the formal parameters of the procedure. The particular values, for example a list of patient-specific symptoms, becomes a data base upon which the abstract procedure operates. Thus, the MYCIN system is an abstract procedure relative to a case library that it can diagnose. The familiar idea is that a general procedure references *relations*, not individuals directly, which is how the data base must be indexed.

The diagnostic procedure of HERACLES is abstract in the sense that metarules mention only non-medical terms, such as *finding* and *hypothesis*, and relations. These terms are variables, instantiated by domain-specific concepts. The metarules are also abstract in the sense that premises are simple logical patterns; backtracking to satisfy variables and indexing the domain representation is left to the interpreter. Specifically, *traditional programming constructs of iteration, variable assignment, and data structure manipulation have been obviated by the use* *of prefix predicate calculus as a procedural specification language.* We have replaced what to do at the Lisp computational level by what is true at the "knowledge level" of the procedure we are describing. Finally, the metarules are abstract characterizations of the procedural knowledge that is stated in a domain-specific, implicit way in MYCIN's rules. In this case, the ordering of specific values, such as the order in which to gather data to confirm a hypothesis, has been abstracted to metarules based on *preference relations* such as *trigger finding* and *necessary precursor.*

What is the nature of HERACLES' abstract procedure? What does it do? Studying it, we find a surprisingly simple pattern. There are only three types of task foci: hypotheses, findings, and domain rules. The purpose of each task is to *select* a new focus (for example, moving from a hypothesis to desirable findings). Metarules do this by *relating a current focus to other findings, hypotheses, and rules in the knowledge base.* Thus, domain-specific orderings ("Does the patient have an infection? Is the infection meningitis?") are replaced by:

- a task, something the program is trying to do with respect to its known findings or believed hypotheses;

- a focus (one or more findings, hypotheses, or rules);

- relations among findings, hypotheses, and rules that enable them to be selected preferentially.

An unexpected effect of stating the diagnostic procedure in this way is that there is no more backward chaining at the domain level. That is, the only reason MYCIN does backward chaining during its diagnostic (history and physical) phase is to accomplish top-down refinement and to apply screening rules. This is an important result. By studying the hundreds of rules in the MYCIN system, factoring out domain relations from control knowledge, we have greatly deepened our understanding of the knowledge encoded in the rules. The previously implicit relations between clauses and between a premise and action that made backchaining necessary are now explicit and used directly for selecting the next primitive action (question to ask, assertion to be made, domain rule to apply).

9.2. The advantages of PPC notation for studying procedures

Below we will consider what the relations of HERACLES reveal about the nature of procedures. But first it is worth pausing to consider the advantages of the prefix predicate calculus notation. None of patterns we have discovered were as evident in the original Lisp version of NEOMYCIN's metarules. When the rules were re-expressed in MRS we stopped thinking so much about programming and started to think about the knowledge we were encoding, viz,

- Most of the "flags" in the program were replaced by unary relations. For example, the Lisp free variable PROBLEM-IDENTIFIED became the statement, (TASK-COMPLETED IDENTIFY-PROBLEM). We discovered that for every flag there was some implicit concept that was being characterized by an implicit relation. Here, we recognized that PROBLEM-IDENTIFIED could be restated as the proposition, "The task IDENTIFY-PROBLEM has completed." Implicitly, the task IDENTIFY-PROBLEM was being characterized by the the relation TASK-COMPLETED. Thus, a new relation was added to the program. This kind of analysis provides a better understanding of the domain and procedural knowledge. It also greatly facilitates program design. For example, context shifting, as required for student modeling, can be handled in a more general way using the predicate calculus notation instead of free variables. Put another way, PPC notation helps us to separate *task-specific control knowledge* from the *implementation of the task interpreter*.

- We found that all domain Lisp properties were relations among findings, hypotheses, and domain rules. The entire domain knowledge base can be viewed as a hierarchical, relational database. Rather than "parameters" and "rules" we began to think in terms of findings, hypotheses, and rules and classifications of these. Similarly, we found that the focus of each task was typed.

- We discovered that all of the predicates in EMYCIN's rule language could be expressed as relations about propositions. For example, (SAME CNTXT PARM VALU), "parm of context is value," could be restated in PPC as (with a partially instantiated example):

```
(AND (BELIEF (<parm> $CNTXT $VALU) $CF)
     ( > $CF 200)).

(AND (BELIEF (SITE CULTURE-1 BLOOD) $CF)
     ( > $CF 200)).
```

  This reveals that EMYCIN's "parameters" are domain-specific relations characterizing various objects (the contexts), and the rule premise functions (e.g., SAME) are relations between a domain proposition and a certainty factor. (See (Clancey, 1985) for further logical analysis of MYCIN's parameters.)

9.3. The relation between classifications and procedures

The PPC notation helps us see patterns: relations among "rule premise functions," among data structures (the IMPLEMENTATION relation used by the compiler), among tasks. The relations in PPC are the patterns that we observe. We observe that different tasks (e.g., explanation, diagnosis, compilation) require that knowledge be used in different ways, as evidenced by the different *second order relations* required by the different procedures we have written:

- The EMYCIN interpreter needs to know only a handful of simple relations. The most important are:

```
(PROMPT $PARM $STRING)
(ASKFIRST $PARM)
(MULTIVALUED $PARM)
(LEGAL-VALUE $PARM $VALU)

(MENTIONS $RULE $PARM $CNTXT)
(CONCLUDES $RULE $PARM $CNTXT)
(ANTECEDENT $RULE)

(ASKED ($PARM $CNTXT $VALU))
(APPLIED $RULE $CNTXT)
(BELIEF ($PARM $CNTXT $VALU) $CF)
```

- The EMYCIN explanation program needs to know how parameters and rules interact during problem solving, e.g., (RULE-FAILED $RULE $CLAUSE).

- HERACLES' classification procedure requires hierarchical and triggering and relations, e.g., (SUBSUMES $FINDING1 $FINDING2).

- The task interpreter needs to know how metarules should be applied, e.g., (TASK-TYPE $TASK $TYPE).

- The rule compiler needs to know the IMPLEMENTATION, whether a relation is a predicate or a function, and whether matching should be exhaustive.

- HERACLES explanation program uses knowledge about how relations can be inferred from one another and belief about what the listener knows (Section 8).

Relations discriminate, they make distinctions that are essential for satisfying procedural objectives, and making distinctions is what control knowledge is all about. It has been known for some time that *classification* is at the core of problem solving (Bruner, et al., 1956). We are constantly faced with choices, and they are arbitrated by classification distinctions that we make about data, operators, beliefs, states, etc.

Relations are a means for indexing domain-specific knowledge: They *select* hypotheses to focus upon, findings to request, and domain inferences that might be made. As such, relations constitute the organization, the access paths, by which strategies bring domain-specific knowledge into play. For example, the metarules given above mention the CHILDOF and SUBSUMES relations. METARULE001 (in Section 2.1) looks for *the children of* the current hypothesis in order to pursue them; METARULE002 looks for *a more general finding* in order to ask for it first.

These relations constitute the language by which the primitive domain concepts (particular findings and disorder hypotheses) are related in a network. *Adding a new strategy often requires adding a new kind of relation to the network.* For example, suppose we desire to pursue common causes of a disorder before serious, but unusual causes. We must partition the causes of any disorder according to this distinction, adding new relations to our language--COMMON-CAUSES and SERIOUS-CAUSES.

Similarly, *the applicability of a strategy depends on the presence of given relations in the domain.* For example, a strategy might give preference to low-cost findings, but in a particular problem domain all findings might be equally easy to attain. Or a given set of strategies might deal with how to search a deep hierarchy of disorders, but in a given domain the hierarchy might be shallow, making the strategies inapplicable. By stating strategies abstractly, we are forced to explicate relations. On this basis we can compare domains with respect to the applicability of strategies, referring to structural properties of the search space.

Lenat has found a similar relationship between heuristics (strategies) and slots (structural relations) in his program for discovering new heuristics (Lenat, 1982). In particular, the ability to reason about heuristics in EURISKO depends on breaking down complex conditions and actions into many smaller slots that the program can inspect and modify selectively. The same observation holds for domain concepts whose representation is refined by the synthesis of new slots (e.g., adding a PRIME-FACTORS slot to every number). The program even reasons *about relations* by creating a new slot that collects relations among entries of an important slot.

More generally, it is interesting to view the domain knowledge base of relations as a *map* for accomplishing some task. The map makes certain distinctions that are useful for the task at hand. For example, if we wanted to understand the conditions for the spread of a blight, we might use maps that reveal soil and climate conditions. In this sense, NEOMYCIN's relational knowledge base is a map, a view of the world, that has practical value for diagnosis.

9.4. The meaning of relations

The need to make new distinctions is tantamount to requiring new kinds of knowledge for each procedure. In representing a procedure, using new relations, we are making new statements about what is true in order to specify what to do (what operators to apply to what focus). That is, we are organizing knowledge so that it is accessible, reviewable, and selectable by procedures. Thus, an existing knowledge base must be continuously structured in richer ways for explanation, student modeling, knowledge acquisition.

But what comes first--the distinctions or the procedure? The procedure itself exists in order to "treat instances" the "right way," where the instances are the arguments to subprocedures. Thus, we classify instances and say instances of a certain class are to be treated a certain way. In so doing, we define classes in terms of their *functional significance*.

For example, we found in NEOMYCIN that certain findings, such as "headache" were triggering new hypotheses, when there was much stronger evidence for hypotheses that accounted for these findings already. In addition, we noticed that the findings in question were so non-specific, that it wasn't even clear that they needed to be explained. Thus, we made a distinction between findings: non-specific (perhaps a sign of nothing treatably abnormal) and "red-flag" (a finding that is so abnormal, it must be explained). With this new relation partitioning the findings, we wrote a metarule, "If a non-specific finding is not already explained by hypotheses suggested by redflag findings, then trigger new hypotheses, if any." Notice that "explained by" is another relation among findings and hypotheses, defined in terms of evidence and subsumption.

The new relations (red-flag, explained-by) are defined in terms of primitive relations (finding, suggests), but what they *mean* is bound up as well in how they are used procedurally, that is their association with diagnostic operators such as making an assertion and asking a question. Hence, the relations are not *natural distinctions* as we might categorize cats and dogs separately; they are functionally defined. The categories exist because they are procedurally useful.

Other relations have a similar procedural interpretation, for example, follow-up question, triggers, general question. The definition of these relations is captured by how they are used:

- this operation, a subtask, is (or is not) applicable to operands (the argument of the task, its focus) of type X (e.g., follow-up question, non-specific finding);

- give preference to operands of type X for this operation (e.g., triggering finding is pursued first in confirming a hypothesis).

This suggests that knowledge to derive the metarules includes being able to derive what operations might be applicable to specific task operands. That is, we must know what operations we want to do (e.g., apply a domain rule to change our belief in a hypothesis) and a *type characterization* of applicable operands. Thus, put the most general way, the situation or premise parts of metarules are searching for operands to which to apply inference operators.

Is there some conceptual definition that could be used to generate these operand distinctions, given other more-primitive, domain relations? Indeed, when we start studying the domain relations (for explanation purposes) we discover that there are implicit implication relations among them (Figure 9-1). For example, a follow-up relation implies that the characterizing finding discriminates the first finding in terms of the *process* that is occurring. At a more general level yet, we find that the follow-up finding *presupposes* the first finding. This suggests that we could define perhaps statistical correlations related to specificity or ability to discriminate hypotheses that could generate these "procedural" relations.

It is apparent that these relations, and even the more primitive relations in HERACLES, such as CAUSES and SUBSUMES, are just a step removed from EMYCIN's domain-specific clause orderings. They specify fairly closely *what to do* or in control terms, *what to infer*. We would like to move closer to *what is true about the world* that makes these relations valid. The knowledge we have stated is in the form of *schemas* indicating typical associations that are valuable for problem solving. In a principled way, we would like to state mathematically (statistically) what characteristics of a case population and what structural (hierarchical) constraints make one finding a better general question than another or should cause a group of findings to trigger a disease. To the extent that these are the relations in which experiential knowledge is stated, we are seeking a model of learning that relates cognitive, social, mathematical and world constraints to produce the distinctions we have recorded in NEOMYCIN's knowledge base (e.g., triggers, follow-up questions, general questions, and red-flag findings). Given such a model, we might also be able to automatically configure a knowledge base given primitive relations from which the procedural relations can be derived.

It is also possible to proceed in the other direction, to view MYCIN as a model of "compiled

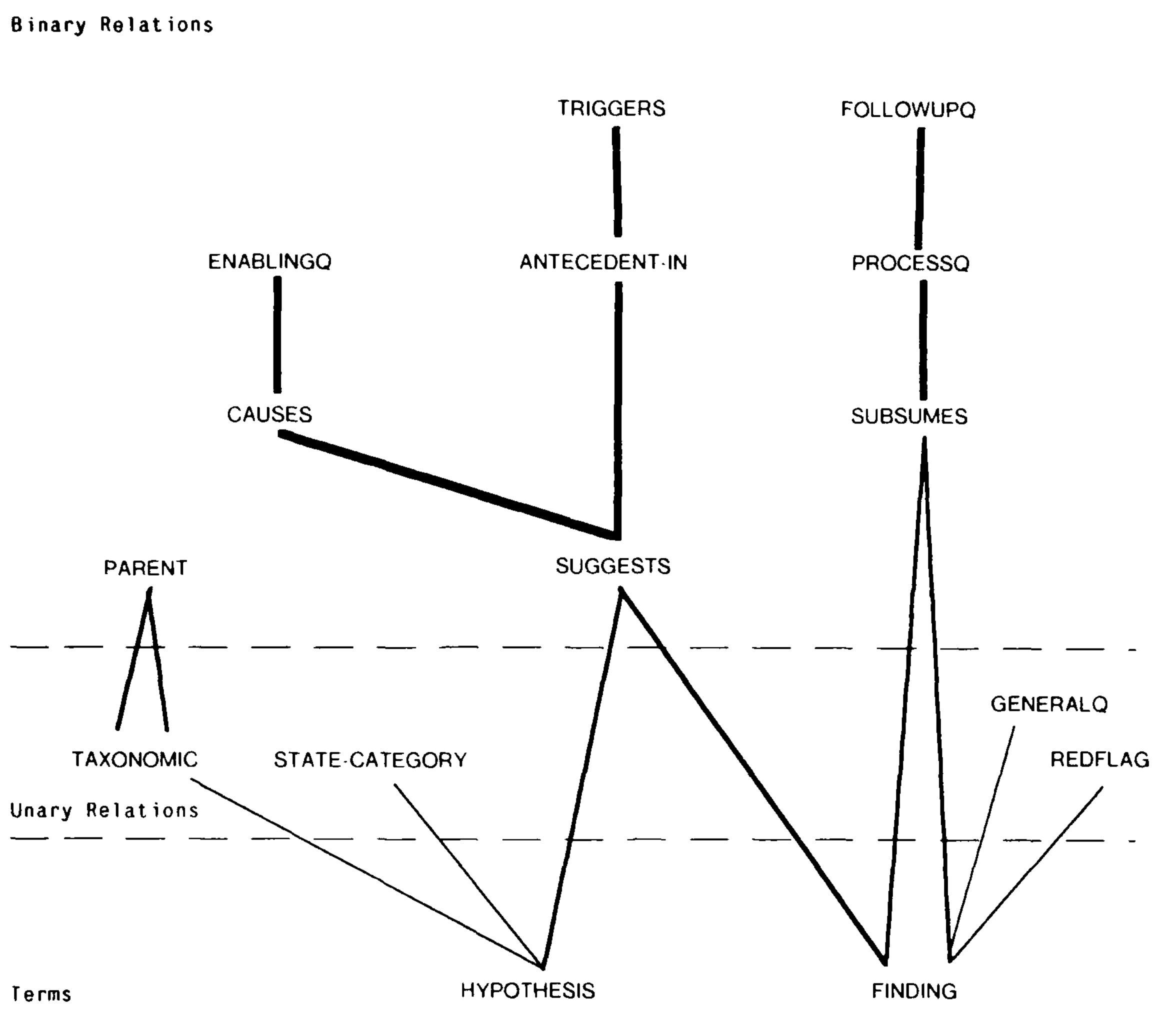


Figure 9-1: Implication relations among HERACLES domain relations [higher relations are defined in terms of lower relations]

knowledge" and NEOMYCIN as a specification of the primitive relations that get instantiated and composed from practice (Anderson et al., 1981, Laird, et al., 1984). While NEOMYCIN's knowledge is computationally equivalent to MYCIN--in terms of focus choice and conclusions--it suggests perhaps a misleading model of learning. It is unlikely that people start only with general principles and instantiate and compose them with experience. The medical knowledge of NEOMYCIN most likely accumulated incrementally, by case experience, not uniformly compiled from explicitly learned generalizations and structural models of the world. Thus, problem solving behavior may appear systematic and principled, but the problem solver may be totally unaware that he is following patterns.

The mechanism of learning may proceed in a way that is analogous to early language learning. We learn how to speak, without being aware of the grammatical rules we are following. Indeed, it is interesting to compare the domain relations to linguistic categories (noun, direct object, demonstrative pronoun, etc.) and the metarules to rules of grammar. Metarules, like rules of grammar, have an abstract, hierarchical nature. Just as the rules of grammar can be used to parse a linguistic utterance, the metarules can be used to parse a sequence of classification actions, as in student modeling, providing a "deep structure" interpretation of behavior (Clancey, 1984b). Is our awareness of the deep structure of language any different from our awareness of the deep structure of diagnosis by classification? Are the learning, representation, and processing mechanisms the same? Answers to these questions are beyond our reach today.

9.5. The difficulties of making a procedure's rationale explicit

It might be objected that for some domains there are no patterns for using knowledge--no abstract procedures--all facts and relations are inseparable from how they will be used. For example, the procedure for confirming any given disorder (more generally, interpreting signals or configuring some device) might be completely situation-specific, so there are no general principles to apply. This would appear to be an unusual kind of domain. We are more familiar with problems in which simple principles can be applied over and over again in many situations.

Teaching and learning are made incredibly difficult if there is no carry-over of procedures from one problem to another. Domains with a strong perceptual component, such as signal interpretation, might be like this. Perceptual skills rely on pattern matching, rather than selective, controlled analysis of data; they are might be poor candidates for representing procedures abstractly.

We also know that in many domains, for efficiency at runtime, procedures have been compiled for solving routine problems. These procedures are written down in the familiar "procedures manuals" for organization management, equipment operation, configuration design, troubleshooting, etc. It is important to recognize that these procedures are based upon domain facts, constraints imposed by causal, temporal, and spatial interactions, problem-solving goals, abstract principles of design, diagnosis, etc. Except where a procedure is arbitrary, there must be some underlying rationale for the selection and ordering of its steps. Knowing this rationale is certainly important for reliably modifying the procedure; such procedures are often just prepared plans that an expert (or a user following a program's advice) may need to adapt to unusual circumstances. At one level, the rationale can be made explicit in terms of an abstract plan with its attendant domain structural relations; a redundant, compiled form can be used for efficient routine problem solving.

In theory, if the rationale for a procedure or prepared plan can be made explicit, a program can reconstruct the procedure from first principles. This approach has two basic difficulties. First, the procedure might have been learned incrementally from case experience. It simply handles problems well; there is no compiled-out theory that can be articulated. This problem arises particularly for skills in which behavior has been shaped over time, or for any problem in which the trace of "lessons" has been poorly recorded. The second difficulty is that constructing a procedure from first principles can involve a great deal of search. Stefik's

(Stefik, 1980) multi-leveled planning regime for constructing MOLGEN experiments testifies to the complexity of the task and the limited capabilities of current programs. In contrast, Friedland's (Friedland, 1979) approach of constructing experiment plans from skeletal, abstract plans trades flexibility for efficiency and resemblance to human solutions. While skeletal plans may sometimes use domain-specific terms, as precompiled abstract procedures they are analogous to HERACLES's tasks.

Importantly, as mentioned in Section 6.2, the *rationale for the abstract plan* itself is not explicit in any of these programs. For example, HERACLES's metarules for a given task might be ordered by preference (alternative methods to accomplish the same operation) or as steps in a procedure. Since the constraints that suggest the given ordering are not explicit, part of the design of the program is still not explicit. For example, the abstract steps of top-down refinement are now stated, but the sense in which they constitute this procedure is not represented. (Why should pursuing siblings of a hypothesis be done before pursuing children?) As another example, the task of "establishing the hypothesis space" by expanding the set of possibilities beyond common, expected causes and then narrowing down in a refinement phase has mathematical, set-theoretic underpinnings that are not explicit in the program. Similarly, Stefik's abstract planning procedure of "least-commitment" is implicit in numeric priorities assigned to plan design operators (Clancey, 1983). Automatically constructing procedures at this high level of abstraction, as opposed to implicitly building them into a program, has been explored very little.

We have considered how relations might be defined in terms of more primitive relations, but is there any hope of formalizing what operators to apply and their applicable operands? Essentially, we would like to represent what a task means, what goal it seeks to accomplish, how the subtasks do this, and why the subtasks and metarules are ordered in a particular way. For example, TEST-HYPOTHESIS is an example of a task in which the metarules capture an ordered set of operand preferences. The goal of the task is to gather evidence that changes belief. This can be accomplished by applying a APPLYRULE operator to a domain rule that makes an assertion about the hypothesis. We could apply any domain rule, but we have some constraints to satisfy, such as seeking the strongest belief first (to focus on the correct diagnosis as quickly as possible). We transform our constraints into preferences that classify possible domain rules. Specifically, we make distinctions about domain rules in terms of findings they mention, the strength of belief in the conclusion, or the justification of the rule. Then we write metarules to select domain rules on the basis of these distinctions. Representing goals and operators as concepts that can be related by the program (for example, so the program can be said in some sense to understand what it means to "gather evidence that changes belief") and relating constraints so they can be reformulated as classifications of domain concepts, are major research problems. ((Clancey, 1984b) lists all of HERACLES' metarules with prosaic descriptions of the constraints they satisfy.)

It is not difficult to find portions of HERACLES' diagnostic procedure whose design is obscure. In most cases, we are still groping for some idea of *what to do*, what the procedure is, and have insufficient experience for deriving a better representation that would express the design of the procedure more clearly. For example, the end conditions previously described encode the procedure implicitly. The rebinding of the list of new findings to bring about depth-first, focused forward-reasoning is also obscure. Probably most disturbing of all, the premises of FINDOUT metarules invoke this task recursively and indirectly through the domain interpreter (should we view them as abstract domain rules, rather than metarules?). At the very least, we now have a good criterion for identifying an implicit procedure: Any encoding that affects the *choice* of operators, operands, or possible inferences that is not accomplished by an explicit classification relation that makes a distinction among them. For example, the relation among metarules is not explicit, so this part of the procedure is implicit (known only to the designer).

For the moment, we are driven by practical needs in writing teaching programs and worry about deficiencies only as they become important to that goal. Indeed, we have found that a good heuristic for representing procedures abstractly is to work with good teachers, for they are most likely to have extracted principles at a level of detail worth teaching to students.

10. Related work

The study of problem solving procedures has played a central role in AI and Cognitive Science. The design of HERACLES has been influenced by a great deal of this previous research.

10.1. Cognitive studies

The idea of formalizing a diagnostic procedure as a separate body of knowledge was inspired by studies of goals and strategies (Greeno, 1976, Schoenfeld, 1981), "deep structure" analyses of problem solving sequences (Brown et al., 1977), psychological studies of medical problem solving (Feltovich et al., 1980, Rubin, 1975, Elstein et al., 1978), and problem space descriptions of behavior (Newell and Simon, 1972). These researchers share the view that *sequences of problem solving steps* could be explained in terms of underlying reasoning principles or strategies. This prompted us to redescribe MYCIN's knowledge in terms of orderly sequences of problem-solving tasks, applied to a medical knowledge base of possible operators, as reported in (Clancey, 1983, Clancey, 1984a, Clancey, 1984b).

10.2. Explicit control knowledge

If one views the cognitive studies as establishing a conceptual base for describing problem solving, research in AI can be characterized as providing a *technological* base for constructing programs. In particular, the work of Davis concerning *metaknowledge*, specifically *metarules*, provided a formalism and way of thinking about program design that strongly affected the design of NEOMYCIN. In turn, the explicit control formalism of PLANNER (Hewitt, 1972) influenced Davis's work. In some sense, this work has all been influenced in a general way by the structured programming philosophy of abstract data types and hierarchical design (Dahl, et al., 1972). Later descriptions of hierarchical planning (Sacerdoti, 1974) inspired the hierarchical task decomposition of the diagnostic procedure of NEOMYCIN.

The idea of a problem solving architecture with explicit bodies of control and domain knowledge has several sources. For example, representing control knowledge abstractly moves us closer to our ideal of specifying to a program *what* problem to solve versus *how* to solve the problem (Feigenbaum, 1977). Dating back at least to McCarthy's Advice Taker (McCarthy, 1960), many researchers have taken the ideal form of a program to be one that could accept new pieces of information incrementally. For a system with this architecture, improving a program is a well-structured process of stating knowledge relations or, separately, HOW the knowledge will be used. This is to be contrasted with doing both simultaneously in an intermingled, redundant way, as in MYCIN's rules.

An analogy can be made with GUIDON (Clancey, 1979) (Clancey, 1982a), whose body of abstract teaching rules makes the program usable with multiple domains. Traditional CAI programs are specific to particular problems (not just problem domains) and have both subject matter expertise and teaching strategies embedded within them. The separation of these in GUIDON, and now the abstract representation of strategies in NEOMYCIN, is part of the logical progression of expert systems research that began with separation of the interpreter from the knowledge base in MYCIN. The trend throughout has been to state domain-specific knowledge more declaratively and to generalize the procedures that control its application.

Probably the earliest, large-scale realization of the ideal separation between knowledge and search control is in HEARSAY, with its well-defined sources of knowledge, multi-leveled abstract description of solutions on a blackboard, and control knowledge for focusing on data and solution elements (Erman, et al., 1980). Several general problem solving architectures have been developed to improve upon this structure, notably including HEARSAY-III (Erman, et al., 1981) and BB1 (Hayes-Roth, 1984). The emphasis in HERACLES has not so much to build a general architecture, but to proceed in a bottom-up way to specify the knowledge and distinctions important for explanation and student modeling. Then, with a developed vocabulary of knowledge relations and a diagnostic procedure, we have proceeded empirically to characterize the meaning of this general control knowledge and how it might be generalized. We find that the complex control requirements of diagnosis require a rich relational vocabulary for relating domain rules, a conclusion that is paralleled in recent extensions to HEARSAY that achieve finer-grained control by making explicit the relations among knowledge sources (Corkill, et al., 1982).

10.3. Logic specification of procedures

A considerable amount of research is specifically concerned with using logic formalisms to represent inference procedures. None of this work was known to us during the development of NEOMYCIN, but the connections are obvious and worth considering.

Perhaps from the most general point of view, this research could be characterized as an attempt to extend logic to make deduction more efficient. For example, under one conception, a data base of facts requires a *smart interpreter* that has knowledge about different deductive methods and different kinds of problems to control its search. AMORD was one of the earliest attempts to represent different inference procedures in logic DEKLEER79. The idea of *partial programs*, continuing in the Advice Taker tradition, is a continuation of this effort to find a suitable language for incrementally improving a program's behavior (Genesereth, 1984).

Similar problems with efficient deduction arise in database systems that combine relational networks with logic programming (e.g., see (Nicolas, 1977)). To conserve space, it is not practical to explicitly store every relation among entities in a database. For example, a database about a population of a country might record just the parents of each person (e.g., (MOTHEROF $CHILD $MOTHER) and (FATHEROF $CHILD $FATHER)). A separate body of *general derivation axioms* is used to retrieve other relations (the *intensional database*). For example, siblings can be computed by the rule:

```
(IF (AND (PERSON $PERSON)
         (MOTHEROF $PERSON $MOTHER)
         (PERSON $PERSON2)
         (MOTHEROF $PERSON2 $MOTHER))
    (SIBLING $PERSON $PERSON2))
```

Such a rule is quite similar to the abstract metarules that NEOMYCIN uses for deducing the presence or absence of findings. NEOMYCIN differs from database systems in that its rules are grouped and controlled to accomplish abstract tasks. Only a few of NEOMYCIN's metarules make inferences about database relations; most invoke other tasks, such as "ask a general question" and "group and differentiate hypotheses." Moreover, the knowledge base contains judgmental rules of evidence for the disorder hypotheses. These differences aside, the analogy is stimulating. It suggests that treating a knowledge base as an object to be inspected, reasoned about, and manipulated by *abstract procedures*--as a database is checked for integrity, queried, and extended by general axioms--is a powerful design principle for building knowledge systems.

10.4. Hybrid systems

HERACLES is a prime example of a *hybrid* system, combining Lisp, predicate calculus, procedural, and rule representations of knowledge. Simple rule and frame systems, in contrast with traditional programs, are valued for their syntactic simplicity, facilitating the writing of interpretation programs for problem solving, explanation, knowledge acquisition, etc. Yet, in the past decade it has become obvious that different representations are advantageous for efficiency and perspicuity. Other researchers have found similar pragmatic advantages to the kind of hybrid design we find in HERACLES. For example, in SOPHIE-III multiple representations are nicely integrated by a common database that allows dependency-directed inference (Brown, 1977, de Kleer, 1984). Rich (Rich, 1982) reports a system that combines predicate calculus with other representations. Furthermore, an initial motivation for MRS was to facilitate multiple representation systems, an early meaning for the acronym. It is now common to find knowledge engineering tools with hybrid architectures, such as the combination of rule and object-oriented programming in LOOPS (Bobrow and Stefik, 1983) and STROBE (Smith, 1984).

Recently, the idea of developing a *knowledge-specification language* that is integrated with implementation data structures by a compiler, as in HERACLES' metarules, has become popular. GLISP programs are "compiled relative to a knowledge base of object descriptions, a form of abstract datatypes" (Novak, 1982). In the context of automatic programming research, the APS language (Cohen and Goldman, 1985) integrates a relational view of Lisp data with predicate calculus, in a manner very similar to HERACLES. STROBE uses an object-oriented language in its control rules, resulting in a syntax somewhere between Lisp and predicate calculus. Preference for these different languages, based on ease of readability, may depend more on familiarity than abstract properties of the formalisms.

Comparing the object-oriented and relational approaches, the ability of an object to respond to a message is analogous to having a relation defined for an instance of a term. Sending a message to an object is analogous to inferring the truth of a proposition when one or more of the terms are bound. The object-oriented view is useful for implementation (e.g., note how our compiler relies on a relation being implemented as a property of the first term), but we believe that a relational view provides a more elegant and revealing perspective for making statements about the knowledge in a system. Metaknowledge takes the form of relations about relations, but statements about messages are neither objects nor messages.

10.5. Rule sets

Rule sets have become popular for hierarchically controlling rules. The control primitives of HERACLES are very similar to those developed contemporaneously in LOOPS. However, this formalism is used in HERACLES for controlling rules that invoke domain rules in a much more complex way, not for organizing the domain rules themselves, as in LOOPS. LOOPS has an agenda mechanism for controlling tasks, which might be advantageous for modeling a process like diagnosis. However, LOOPS lacks the pattern-matching language that we have found to be essential for the indexing and selection operations of an abstract inference procedure. Georgeff (Georgeff, 1982) proposes a framework for procedural control of production systems that bears some resemblance to HERACLES rule sets, but again, he organizes domain rules directly. The task and rule set idea of HERACLES has been applied in STROBE, but here with some abstract metarules for controlling search (Young, et al., 1985).

10.6. Explanation

Aside from its obvious origin in the original explanation research of Shortliffe and Davis, HERACLES research has developed in parallel with the work of Swartout, with complementary conclusions and methodology. An interesting distinction is that Swartout has focused on explaining numerical procedures (how a drug's dosage is computed), while we have focused on the more general procedures of classification reasoning. Consequently, we have easily stated our procedures abstractly, while it is less obvious that a procedure that is more like a recipe for doing something has a meaningful or useful generalization. The level of abstraction aside, both XPLAIN and HERACLES achieve explanations through the separate and explicit representation of domain facts and the procedure for their inclusion and ordering in problem solving. Furthermore, Swartout demonstrated that an automatic programming approach, as difficult as it first seemed, was a natural, direct way to ensure that the program had knowledge of its own design (Swartout, 1981). That is, providing complete explanations means understanding the design well enough to derive the procedures yourself.

As discussed in Section 9.5, the search process of diagnosis, particularly for focus of attention, is so complex that we don't know exactly what the program should do, let alone justify what it currently does in any principled way. However, we believe that the methodology of constructing the program from a specification would be good approach for studying and making more explicit the constraints that lie behind the tasks and metarules.

10.7. The meaning of procedures

This research has clarified for us that there is a useful distinction at a "knowledge level" between facts (what we declare to be true) and procedures (sequential descriptions of what we do). Pursuing our ideal of making explicit the rationale that lies behind every piece of knowledge in our programs, we considered in Section 9 the meaning of HERACLES' inference procedure. This kind of analysis, particularly the search for unspecified constraints, has been strongly influenced by the work of van Lehn and Brown (VanLehn and Brown, 1979), specifically in their very detailed analysis of mathematical procedures. In this restricted domain they have derived alternative procedures from constraints that define operators and the representations being manipulated. This level of precision remains an ideal that directs continuing HERACLES research. Brown's preliminary study of the *semantics of procedures* (Brown, et al., 1982b) reveals a combination of orthogonal constraints very similar to what we have discovered in analyzing HERACLES metarules.

11. Summary of advantages

The advantages of representing control knowledge abstractly can be summarized according to engineering, scientific, and practical benefits:

- **Engineering.**
    - The explicit design is easier to debug and modify. Hierarchical relations among findings and hypotheses and search strategies are no longer procedurally embedded in rules.
    - Knowledge is represented more generally, so we get more performance from less system-building effort. We don't need to specify every situation in which a given fact should be used.
    - The body of abstract control knowledge can be applied to other problems, constituting the basis of a generic system, for example, a tool for building consultation programs that do diagnosis.
- **Science.** Factoring out control knowledge from domain knowledge provides a basis for studying the nature of strategies. Patterns become clear, revealing, for example, the underlying structural bases for backward chaining. Comparisons between domains can be made according to whether a given relation exists or a strategy can be applied.
- **Practice.**
    - A considerable savings in storage is achieved if abstract strategies are available for solving problems. Domain-specific procedures for dealing with all possible situations needn't be compiled in advance.
    - Explanations can be more detailed, down to the level of abstract relations and strategies, so the program can be evaluated more thoroughly and used more responsibly.

- o Because strategies are stated abstractly, the program can recognize the application of a particular strategy in different situations. This provides a basis for explanation by analogy, as well as recognizing plans during knowledge acquisition or student modelling.

There is certainly an initial cost to stating procedures abstractly, whose benefit is unlikely to be realized if no explanation facility is desired, only the original designers maintain or modify the knowledge base, or there is no desire to build a generic system. But even this argument is dubitable: a knowledge base with embedded strategies can appear cryptic to even the original designers after it has been left aside for a few months. The quality of a knowledge base depends not only on how well it solves problems, but also how on easily its design allows it to be maintained. Easy maintenance--the capability to reliably modify a knowledge base without extensive reprogramming--is important for several reasons:

- Knowledge-based programs are built incrementally, based on many trials, so modification is continually required, including updates based on improved expertise (it was very difficult to add knowledge about new infections to MYCIN because of the implicit search procedure in existing rules).
- A knowledge base is a repository that other researchers and users may wish to build upon years later;
- A client receiving a knowledge base constructed for him may wish to correct and extend it without the assistance of the original designers.
- Also, anyone intending to build more than one system will benefit from expressing knowledge as generally as possible so that lessons about structure and strategy can speed up the building of new systems.

A knowledge base is like a traditional program in that maintaining it requires having a good understanding of the underlying design. Problems encountered in understanding traditional programs--poorly-structured code, implicit side-effects, and inadequate documentation--carry over to knowledge base maintenance. The architecture advocated here--statement of inference procedures abstractly, in a well-structured language -avoids procedural embedding, forcing the knowledge engineer to state domain knowledge in a well-structured way, so it can be flexibly indexed by his procedures.

12. Conclusions

This research began with the conjecture that the procedure for diagnosis could be separated from the medical knowledge base, and that this would offer advantages for explanation, student modeling, and knowledge engineering. NEOMYCIN, its generalization to HERACLES, and the associated prototype programs (Sections 7 and 8) demonstrate that the separation and explicit representation of procedural knowledge has merit and is possible. The architecture is convenient, and all indications are that it provides a good basis for further investigation in the nature of procedural knowledge and its use for different purposes.

It is worth noting that our methodology, driven by failures of our programs to meet behavioral objectives, has been very valuable. We select a level of explicitness desirable in our knowledge bases through by using them in diverse programs (e.g., explanation, modeling). Then we note the limitations for interpreting this knowledge representation, particularly what distinctions need to be made explicit so the procedural knowledge (for diagnosis, compiling, explaining, modeling) can make intelligent choices among alternatives. This continues our approach begun in MYCIN of developing a language, building specific programs for difficult problems, studying the knowledge we have collected, and then repeating the cycle. Our conjectures about problem solving architecture gain credence through experimentation: a given task/metarule is applied in multiple contexts in a given problem, interpreted for different behaviors (e.g., explanation and modeling), and carried over to other domains in new HERACLES systems. In this way, we exploit the idea of abstract control knowledge and gain confidence in the generality of our results. Rather than just building one "expert system" after another, we are developing theories about kinds of knowledge and methods for solving problems.

Critiquing HERACLES today, what does it need to be a more general, useful system for modeling human classification problem solving? We would incorporate, at least: KL-ONE style definitions of concepts and general subsumption algorithm (Schmolze and Lipkis, 1983); hierarchies as lattices allowing multiple parents; conceptual graph definitions of relations SOWA84; an agenda mechanism for tasks; and dependency reasoning for belief maintenance (de Kleer, 1984). Each of these components is getting so complex, it is unlikely that the community of AI researchers can afford much longer to redundantly re-implement them in every architecture. At some point, we may need to develop interface languages that allow us to share modules, permitting individual decisions about the level of implementation, domain structures, and control, with compilation used to piece together hybrid systems like HERACLES. If this is to be so, the kind of separation and explicit representation of different kinds of knowledge, organized around a relational language, as we advocate, may become an essential principle for building knowledge systems.

I. Metarule relational language

The relations used in HERACLES' metarules and rules for metarule premise relations are listed here. They are categorized as domain, dynamic belief, dynamic search or focus bookkeeping, and computational. Inverses (e.g., causes and caused-by) are omitted here, but included in the knowledge base. The important primitive terms are $PARM, $RULE, $CF, and $CNTXT. All other terms and relations are defined in these terms. Indentation indicates hierarchical definition of new terms. For example, a nonspecific finding is a kind of finding. These relations are generally implemented as Lisp structures; the dynamic belief relations are all implemented as Lisp functions.

Domain Relations Pertaining to Findings and Hypotheses

```
(FINDING $PARM)
        (SOFT-DATA $FINDING)
        (HARD-DATA $FINDING)
        (NONSPECIFIC $FINDING)
        (REDFLAG $FINDING)
(HYPOTHESIS $PARM)
        (STATE-CATEGORY $HYP)
        (TAXONOMIC $HYP)
                (PARENTOF $TAXPARM $PARENT)
                (COMPLEX $TAXPARM)

(CAUSES $HYP1 $HYP2)
(SUBSUMES $FINDING1 $FINDING2)
(PROCESSQ $FINDING1 $FINDING2)
(CLARIFYQ $FINDING1 $FINDING2)
(SOURCE $FINDING1 $FINDING2)
(SCREENS $FINDING1 $FINDING2)
(PROCESS-FEATURES $HYP $SLOT $VAL $FINDING)

(ALWAYS-SPECIFY $FINDING)
(ASKFIRST $FINDING)
(PROMPT $FINDING $VAL)

(BOOLEAN $PARM)
(MULTIVALUED $PARM)
(TABLE $BLOCKPARM $FINDING)

(ENABLINGQ $HYP $FINDING)
(SUGGESTS $PARM $HYP)
(TRIGGERS $PARM $HYP)
```

Domain Relations Pertaining to Rules

(ANTECEDENT-IN $FINDING $RULE)
(APPLICABLE? $RULE $CNTXT $FLG)
(EVIDENCEFOR? $PARM $HYP $RULE $CF)
(COMMONCASERULES $HYP $RULE)
(UNUSUALCASERULES $HYP $RULE)

(PREMISE $RULE $VAL)
(ACTION $RULE $VAL)
(ANTECEDENT $RULE)
(TRIGGER $RULE)
(SCREEN $RULE)

Dynamic Belief Relations

(BELIEF $HYP $CF)
(CUMCF-VALUE $HYP $CF)
(MAX-CONSIDERED-HYP-CUMCF $CF)

(PREVIEW $CNTXT $RULE)

(DEFINITE $CNTXT $PARM)
(DEFIS $CNTXT $PARM $VALUE)
(DEFNOT $CNTXT $PARM $VALUE)
(NOTKNOWN $CNTXT $PARM)
(SAME $CNTXT $PARM $VALUE $CF)
(SAMEP $CNTXT $PARM)

Dynamic Search or Focus Relations

(CONSIDERED $HYP)
(DESCENDENTS-EXPLORED $TAXPARM)
(EXPLORED $TAXPARM)
(PARENTS-EXPLORED $TAXPARM)
(PURSUED $HYP)
(REFINED $HYP)
(REFINED-COMPLEX $HYP)

(APPLIEDTOP $RULE $CNTXT)
(DONTASKP $CNTXT $PARM)
(TRACEDP $CNTXT $PARM)

(CLARIFIED $FINDING)
(ELABORATED $FINDING)
(SPECIFICS-REQUESTED $FINDING)
(SUBSUMPTION-CONCLUDED $FINDING)
(USERSUPPLIED $FINDING)

(TASK-COMPLETED $TASK)

Dynamic Relations with Changing Values

(CURFOCUS $HYP)
(DIFFERENTIAL $HYP)
(NEW.DIFFERENTIAL)
(WIDER.DIFFERENTIAL)
(DIFFERENTIAL.COMPACT)

(NEXT-HARD-DATAQ $FINDING)
(NEW.DATA $FINDING)
(PARTPROC.DATA $FINDING)

Computational Relations

(ABS $ARG $RESULT)
(CFCOMBINE $CF1 $CF2 $RESULT)
(EQ $ARG1 $ARG2)
(GREATERP $ARG1 $ARG2)
(LESSP $ARG1 $ARG2)
(MINUS $ARG1 $ARG2 $RESULT)
(MINUSP $ARG)
(NULL $ARG)
(TIMES $ARG1 $ARG2 $RESULT)

(FIRST-ONE $LIST $RESULT)
(LENGTH $LIST $RESULT)
(MAKESET $PROP $COLLECTVAR $RESULTVAR)
(MEMBER $MEM $SET])
(SINGLETON? $LIST)

(PREDICATE $REL)
(IMPLEMENTATION $REL $VAL)
(MULTIPLEMATCH $REL)

(UNIFY $PATTERN $FACT)

Metarule Premise Relations (Composites)

(ACTIVE.HYP? $HYP)
(ALWAYS-SPECIFY? $FINDING)
(ANTECEDENT.RULES? $PARM $RULE)
(ANY.ANCESTOR? $HYP1 $HYP2)
(BESTCOMPETITOR $CURRENTHYP $BETTERHYP $BHCF)
(BESTHYP $HYP)
(CHILDOF $HYP $CHILD)
(CLARIFY.QUESTIONS $FINDING $PROCPARM)
(DIFF.EXPLAINED $FINDING)
(DIFF.NOTPARENTS-EXPLORED?)
(DIFF.NOTPURSUED?)
(ELIGIBLECHILD)
(ENABLING.QUESTIONS $HYP $RULE)
(EXPLAINEDBY $FINDING $HYP)
(EXPLORE.CHILD? $HYP $H)
(EXPLORE.HYP? $HYP)
(EXPLORE.SIBLING? $OLDFOCUS $HYP)
(NEXTGENERALQ? $FOCUSQ)
(PARTPROC.NOTELABORATED? $FINDING)
(PARTPROC.SUGGESTRULES? $PARM $RULE)
(POP-FINDING)
(POP-HYPOTHESIS)
(POP-REDFLAG-FINDING $FOCUSFINDING)
(PROCESS-QUESTIONS? $PARM $PROCTYPEPARM)
(REFINABLE? $HYP)
(REFINABLENODE? $OLDFOCUS $FOCUSCHILD)
(REMAINING.QUESTIONS $HYP $RULE)
(SINGLE.TOPCAUSE $FOCUS)
(SOURCEOF $PARM $SOURCE)
(STRONG-COMPETITOR? $CURRENTHYP $BESTCOMP)
(SUBSUMPTION.SUBTRACED $CNTXT $PARM)
(SUBSUMPTION.SUPERFO $CNTXT $PARM)
(SUBSUMPTION.SUPERTRACED $CNTXT $PARM)
(SUBSUMPTION.SUPERUNK $CNTXT $PARM)
(SUGGESTRULES? $PARM $RULE)
(SUPERS.NOTRACED $PARM $SUPERPARM)
(TAXANCESTOR $HYP1 $HYP2)
(TAXREFINE? $HYP)
(TOP.UNCONFIRMED? $ANCESTOR)

```
(TOPUNCON $ANCESTOR $HYP)
(TRIGGERQ $HYP $RULE)
(TRIGGERS? $FINDING $RULE)
(UNAPPLIED? $RULE)
(UNCLARIFIED-FINDING $NEW.DATA $FINDING)
(UNEXPLOREDIFF.COMPACT? $HYP)
(UPDATE.DIFF.RULES? $FINDING $RULE)
(WAITINGEVIDRULES? $HYP $RULE)
(WEAK.EVIDENCE.ONLY? $HYP)
```

Acknowledgments and historical notes

I originally designed and implemented NEOMYCIN in November 1980, following 10 months of studying MYCIN with Reed Letsinger and the late Timothy Beckett, MD. Reed spent a year extending the original metarules and knowledge base as part of his MSAI practicum. The program was first presented in Pittsburgh at the ONR Annual Contractors' Meeting for research in instructional systems in January 1981, and then at IJCAI in August (Clancey, 1981).

Conrad Bock designed and implemented the procedural attachment mechanism and translated the original functions into MRS rules in the summer of 1982. Without Conrad's outstanding ability to translate the original Lisp metarules into MRS many of the results in this paper, particularly the study of procedural relations and the PPC-based explanation program, would not have been possible. Large portions of the appendices describing the MRS/NEOMYCIN implementation were written jointly by us, and appear in (Clancey and Bock, 1982). We thank Avron Barr, Greg Cooper, Lee Erman, and Diane Kanerva for commenting on an earlier version of this material.

David Wilkins (a student visiting Stanford from Michigan) brought Conrad's system up-to-date in the summer of 1983, and converted the metarules to replace the impractically slow deliberation/action loop, metacontrol rules, and stack mechanism by a modified task interpreter written in Lisp which invoked MRS only for evaluating metarule premises. This was also too slow, so we reverted to the original Lisp metarules.

In the fall of 1984, we were finally ready to use the MRS representation for explanation. Diane Warner Hasling helped convert Conrad's and David's MRS expressions into Lisp property list format, and wrote code to analyze the relations. This led to a set of rules that was clean enough to compile; I wrote the compiler.

As acknowledged in the individual sections, Diane Hasling, Bob London, Mark Richer, and Tim Thompson have made major contributions to the explanation and modeling routines and Heracles over the past 5 years. We thank Bruce Buchanan for serving as general scientific advisor and political councilor to the project.

All of the programs described here currently run in InterLisp-D on Xerox 1000 Series machines, connected to a VAX-UNIX file server. Computational resources have been provided by the SUMEX-AIM facility (NIH grant RR00785).

This research has been supported in part by ONR and ARI Contract N00014-79C-0302. As of March 1985, the research is supported in part by the Josiah Macy, Jr. Foundation, award B852005.

References

Aikins J. S. *Representation of control knowledge in expert systems*, in *Proceedings of the First AAAI*, pages 121-123, 1980.

Anderson, J. R., Greeno, J. G., Kline, P. J., and Neves, D. M. Acquisition of problem-solving skill. In Anderson (editor), *Cognitive Skills and their Acquisition*, . Lawrence Erlbaum Associates, Hillsdale, 1981.

Bobrow, D. G. and Stefik, M. The LOOPS Manual. (Xerox PARC).

Brown, J. S. *Remarks on building expert systems (Reports of panel on applications of artificial intelligence)*, in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 994-1005, 1977.

Brown, J. S. *Process versus product--a perspective on tools for communal and informal electronic learning*, in *Education in the Electronic Age, proceedings of a conference sponsored by the Educational Broadcasting Corporation, WNET/Thirteen*, July, 1983.

Brown, J. S., Collins, A., and Harris, G. Artificial intelligence and learning strategies. In O'Neill (editor), *Learning Strategies*, . Academic Press, New York, 1977.

Brown, J. S., Burton, R. R., and de Kleer, J. Pedagogical, natural language, and knowledge engineering techniques in SOPHIE I, II, and III. In D. Sleeman and J. S. Brown (editors), *Intelligent Tutoring Systems*, pages 227-282. Academic Press, London, 1982.

Brown, J. S., Moran, T. P., and Williams, M. D. The semantics of procedures: A cognitive basis for training procedural skills for complex system maintenance. (Xerox Corporation, CIS working paper, November 1982).

Bruner, J. S., Goodnow, J. J., and Austin, G. A. *A Study of Thinking*. New York: John Wiley & Sons, Inc. 1956.

Clancey, W. J. Tutoring rules for guiding a case method dialogue. *The International Journal of Man-Machine Studies*, 1979, *11*, 25-49. (Also in Sleeman and Brown (editors), *Intelligent Tutoring Systems*, Academic Press, 1982).

Clancey, W. J. and Letsinger, R. *NEOMYCIN: Reconfiguring a rule-based expert system for application to teaching*, in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 829-836, 1981. (Revised version in Clancey and Shortliffe (editors), *Readings in medical artificial intelligence: The first decade*, Addison-Wesley, 1984).

Clancey, W. J. GUIDON. In Barr and Feigenbaum (editors), *The Handbook of Artificial Intelligence*, chapter Applications-oriented AI research: Education. William Kaufmann, Inc., Los Altos, 1982.

Clancey, W. J. Applications-oriented AI research: Education. In Barr and Feigenbaum (editors), *The Handbook of Artificial Intelligence*, . William Kaufmann, Inc., Los Altos, 1982.

Clancey, W. J. The epistemology of a rule-based expert system: A framework for explanation. *Artificial Intelligence*, 1983, *20(3)*, 215-251.

Clancey, W.J. Methodology for Building an Intelligent Tutoring System. In Kintsch, Miller, and Polson (editors), *Method and Tactics in Cognitive Science*, pages 51-83. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1984.

Clancey, W. J. *Acquiring, representing, and evaluating a competence model of diagnosis.* HPP Memo 84-2, Stanford University, February 1984. (To appear in Chi, Glaser, and Farr (Eds.), *The Nature of Expertise*, in preparation.).

Clancey, W. J. *Heuristic Classification.* Working Paper, KSL 85-5, Stanford University, March 1985.

Clancey, W. J. and Bock, C. *MRS/NEOMYCIN: Representing metacontrol in predicate calculus.* HPP Memo 82-31, Stanford University, November 1982.

Cohen, D. and Goldman, N. Efficient compilation of virtual database specifications. (U.S.C. Information Sciences Institute).

Corkill, D. D., Lesser, V. R., and Hudlicka, E. *Unifying data-directed and goal-directed control: An example and experiments*, in *Proceedings of the National Conference on Artificial Intelligence*, pages 143-147, August, 1982.

Dahl, O. J., Dijkstra, E. W., and Hoare, C. A. R. *Structured Programming.* New York: Academic Press 1972.

Davis R. *Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases.* HPP Memo 76-7 and AI Memo 283, Stanford University, July 1976.

Davis, R. Meta-rules: reasoning about control. *Artificial Intelligence*, 1980, *15*, 179-222.

Davis, R., Buchanan, B., and Shortliffe, E. H. Production rules as a representation for a knowledge-base consultation program. *Journal of Artificial Intelligence*, 1977, *8(1)*, 15-45.

de Kleer, J. Qualitative and quantitative reasoning in classical mechanics. In P. H. Winston and R. H. Brown (editors), *Artificial Intelligence: An MIT Perspective*, pages 9-30. The MIT Press, Cambridge, 1979.

de Kleer, J. *Choices without backtracking*, in *Proceedings of the National Conference on Artificial Intelligence*, pages 79-85, August, 1984.

Duda, R. O. and Shortliffe, E. H. Expert systems research. *Science*, 1983, *220*, 261-268.

Elstein, A. S., Shulman, L. S., and Sprafka, S. A. *Medical problem solving: An analysis of clinical reasoning.* Cambridge: Harvard University Press 1978.

Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, D. R. The Hearsay-II speech understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 1980, *12(2)*, 213-253.

Erman, L. D., London, P. E., and Fickas, S. F. *The design and example use of Hearsay-III*, in *Proceedings Seventh International Joint Conference on Artificial Intelligence*, pages 409-415, August, 1981.

Feigenbaum, E. A. *The art of artificial intelligence: I. Themes and case studies of knowledge engineering*, in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 1014-1029, August, 1977.

Feltovich, P. J., Johnson, P. E., Moller, J. H., and Swanson, D. B. The role and development of medical knowledge in diagnostic expertise. Presented at the 1980 Annual meeting of the

American Educational Research Association; in Clancey and Shortliffe (editors), *Readings in medical artificial intelligence: The first decade*, Addison-Wesley, 1984).

Friedland, P. E. *Knowledge-based experiment design in molecular genetics*. Technical Report STAN-CS-79-771, Stanford University, October 1979.

Genesereth, M. R. *An overview of meta-level architecture*, in *Proceedings of The National Conference on Artificial Intelligence*, pages 119-124, August, 1983.

Genesereth, M. R. *Partial programs*. HPP Memo 84-1, Stanford University, November 1984.

Georgeff, M. P. Procedural Control in Production Systems. *Artificial Intelligence*, 1982, *(18)*, 175-201.

Greeno, J. G. Cognitive objectives of instruction: Theory of knowledge for solving problems and answering questions. In Klahr (editor), *Studies in Syntax*, . Erlbaum Associates, Hillsdale, 1976.

Hasling, D. W., Clancey, W. J., Rennels, G. R. Strategic explanations for a diagnostic consultation system. *The International Journal of Man-Machine Studies*, 1984, *20(1)*, 3-19.

Hayes-Roth, B. *BB1: An architecture for blackboard systems that control, explain, and learn about their own behavior*. HPP Memo 84-16, Stanford University, December 1984.

Hewitt, C. E. *Description and theoretical analysis (using schemata) of PLANNER: a language for proving theorems and manipulating models in a robot*. Technical Report 258, MIT AI Laboratory, 1972.

Johnson, P. E. What kind of expert should a system be? *The Journal of Medicine and Philosophy*, 1983, *8*, 77-97.

Laird, J. E. *Universal Subgoaling*. PhD thesis, Computer Science Department, Carnegie-Mellon University, 1983.

Laird, J. E., Rosenbloom, P. S., and Newell, A. *Towards chunking as a general learning mechanism*, in *Proceedings of the National Conference on Artificial Intelligence*, pages 188-192, August, 1984.

Lenat, D. B. The nature of heuristics. *Artificial Intelligence*, 1982, *19(2)*, 189-249.

London, B. and Clancey, W. J. *Plan recognition strategies in student modeling: prediction and description*, in *Proceedings of the Second AAAI*, pages 335-338, 1982.

McCarthy, J. *Programs with common sense*, in *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 403-410, 1960. (Reprinted in Semantic Information Processing, M. Minsky (Ed), MIT Press, Cambridge, 1968).

Miller, J. *States of Mind*. New York: Pantheon Books 1983.

Moore, R. C. *The role of logic in knowledge representation and commonsense reasoning*, in *Proceedings of the National Conference on Artificial Intelligence*, pages 428-433, August, 1982.

Genesereth, M. R., Greiner, R., and Smith, D. E. *MRS Dictionary*. MEMO HPP-82-24, Stanford University, 1982.

Newell, A. and Simon, H. A. *Human problem solving.* Englewood Cliffs: Prentice-Hall 1972.

Nicolas, J. M. and Gallaire, H. Data base: Theory vs. interpretation. In H. Gallaire and J. Minker (editors), *Logic and data bases*, pages 33-54. Plenum Press, New York, 1977.

Novak, G. S., Jr. *Data abstraction in GLISP.* HPP Memo 82-34, Stanford University, 1982.

Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas.* : Basic Books, Inc. 1980.

Rich, C. *Knowledge Representation Languages and Predicate Calculus*, in *Proceedings of the National Conference on Artificial Intelligence*, pages 193-196, AAAI, 1982.

Richer, M. and Clancey, W. J. GUIDON WATCH: A graphic interface for browsing and viewing a knowledge-based system. (Submitted to *IEEE Computer Graphics and Applications*).

Rubin, A. D. *Hypothesis formation and evaluation in medical diagnosis.* Technical Report AI-TR-316, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, January 1975.

Rumelhart, D. E. and Norman, D. A. *Representation in memory.* Technical Report CHIP-116, Center for Human Information Processing, University of California, June 1983.

Sacerdoti, E. D. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 1974, *5(2)*, 115-135.

Schmolze, J. G. and Lipkis, T. A. *Classification in the KL-ONE knowledge representation system*, in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 330-332, August, 1983.

Schoenfeld, A. H. *Episodes and executive decisions in mathematical problem solving.* Technical Report, Hamilton College, Mathematics Department, 1981. Presented at the 1981 AERA Annual Meeting, April 1981.

Shortliffe, E. H. *Computer-based medical consultations: MYCIN.* New York: Elsevier 1976.

Smith, R. G. Programming with rules in Strobe. (Schlumberger-Doll Research).

Stefik, M. *Planning with constraints.* STAN-CS-80-784 and HPP Memo 80-2, Stanford University, January 1980.

Swartout W. R. *Explaining and justifying in expert consulting programs*, in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 815-823, August, 1981.

Szolovits, P., Hawkinson, L., and Martin, W. A. An overview of OWL, a language for knowledge representation. In G. Rahmstort and M. Ferguson (editor), *Proceedings of the Workshop on Natural Language Interaction with Databases*, pages 140-156. International Institute for Applied Systems Analysis, Schloss Laxenburg, Austria, 1978. (also appeared as MIT Techreport, TM-86, June 1977).

VanLehn, K. and Brown, J. S. Planning nets: a representation for formalizing analogies and semantic models of procedural skills. In R. E. Snow, Frederico, P. A., and Montague, W. E. (editor), *Aptitude learning and instruction: Cognitive process and analyses*, . Lawrence Erlbaum Associates, Hillsdale, 1979.

Young, R. L., Smith, R. G., and Pirie, G. Organizing a knowledge-based system for

independence. (Schlumberger-Doll Research).

Personnel Analysis Division,
AF/MPXA
5C360, The Pentagon
Washington, DC 20330

Air Force Human Resources Lab
AFHRL/MPD
Brooks AFB, TX 78235

AFOSR,
Life Sciences Directorate
Bolling Air Force Base
Washington, DC 20332

Dr. Robert Ahlers
Code N711
Human Factors Laboratory
Naval Training Systems Center
Orlando, FL 32813

Dr. Ed Aiken
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. William E. Alley
AFHRL/MOT
Brooks AFB, TX 78235

Dr. Earl A. Alluisi
HQ, AFHRL (AFSC)
Brooks AFB, TX 78235

Dr. John R. Anderson
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

Dr. Nancy S. Anderson
Department of Psychology
University of Maryland
College Park, MD 20742

Dr. Steve Andriole
George Mason University
School of Information
Technology & Engineering
4400 University Drive
Fairfax, VA 22030

Dr. John Annett
University of Warwick
Department of Psychology
Coventry CV4 7AJ
ENGLAND

Dr. Phipps Arabie
University of Illinois
Department of Psychology
603 E. Daniel St.
Champaign, IL 61820

Technical Director, ARI
5001 Eisenhower Avenue
Alexandria, VA 22333

Special Assistant for Projec
OASN(M&RA)
5D800, The Pentagon
Washington, DC 20350

Dr. Michael Atwood
ITT - Programming
1000 Oronoque Lane
Stratford, CT 06497

Dr. Patricia Baggett
University of Colorado
Department of Psychology
Box 345
Boulder, CO 80309

Dr. Eva L. Baker
UCLA Center for the Study
of Evaluation
145 Moore Hall
University of California
Los Angeles, CA 90024

Dr. James D. Baker
Director of Automation
Allen Corporation of America
401 Wythe Street
Alexandria, VA 22314

Dr. Meryl S. Baker
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Donald E. Bamber
Code 71
Navy Personnel R & D Center
San Diego, CA 92152-6800

Mr. J. Barber
HQS, Department of the Army
DAPE-ZBR
Washington, DC 20310

Capt. J. Jean Belanger
Training Development Division
Canadian Forces Training System
CFTSHQ, CFB Trenton
Astra, Ontario, KOK
CANADA

CDR Robert J. Biersner, USN
Naval Biodynamics Laboratory
P. O. Box 29407
New Orleans, LA 70189

Dr. Menucha Birenbaum
School of Education
Tel Aviv University
Tel Aviv, Ramat Aviv 69978
ISRAEL

Dr. Werner P. Birke
Personalstammamt der Bundeswehr
Kolner Strasse 262
D-5000 Koeln 90
FEDERAL REPUBLIC OF GERMANY

Dr. Gautam Biswas
Department of Computer Science
University of South Carolina
Columbia, SC 29208

Dr. John Black
College, Columbia Univ.
525 West 121st Street
New York, NY 10027

Dr. Arthur S. Blaiwes
Code N711
Naval Training Systems Center
Orlando, FL 32813

Dr. Robert Blanchard
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Jeff Bonar
Learning R&D Center
University of Pittsburgh
Pittsburgh, PA 15260

Dr. J. C. Boudreaux
Center for Manufacturing
Engineering
National Bureau of Standards
Gaithersburg, MD 20899

Dr. Gordon H. Bower
Department of Psychology
Stanford University
Stanford, CA 94306

Dr. Richard Braby
NTSC Code 10
Orlando, FL 32751

Dr. Robert Breaux
Code N-095R
Naval Training Systems Cente
Orlando, FL 32813

Dr. John S. Brown
XEROX Palo Alto Research
Center
3333 Coyote Road
Palo Alto, CA 94304

Dr. Bruce Buchanan
Computer Science Department
Stanford University
Stanford, CA 94305

Dr. Howard Burrows
National Institute of Health
Bldg. 10, Room 5C-106
Bethesda, MD 20892

Dr. Patricia A. Butler
OERI
555 New Jersey Ave., NW
Washington, DC 20208

Dr. Tom Cafferty
Dept. of Psychology
University of South Carolina
Columbia, SC 29208

Dr. Robert Calfee
School of Education
Stanford University
Stanford, CA 94305

Joanne Capper
Center for Research into Practice
1718 Connecticut Ave., N.W.
Washington, DC 20009

Dr. Jaime Carbonell
Carnegie-Mellon University
Department of Psychology
Pittsburgh, PA 15213

Mr. James W. Carey
Commandant (G-PTE)
U.S. Coast Guard
2100 Second Street, S.W.
Washington, DC 20593

Dr. Susan Carey
Harvard Graduate School of
Education
337 Gutman Library
Appian Way
Cambridge, MA 02138

Dr. Pat Carpenter
Carnegie-Mellon University
Department of Psychology
Pittsburgh, PA 15213

Prof. John W. Carr III
Department of Computer Science
Moore School of
Electrical Engineering
University of Pennsylvania
Philadelphia, PA 19104

Dr. Robert Carroll
OP 01B7
Washington, DC 20370

LCDR Robert Carter
Office of the Chief
of Naval Operations
OP-01B
Pentagon
Washington, DC 20350-2000

Dr. Fred Chang
Navy Personnel R&D Center
Code 51
San Diego, CA 92152-6800

Dr. Alphonse Chapanis
8415 Bellona Lane
Suite 210
Buxton Towers
Baltimore, MD 21204

Dr. Davida Charney
Department of Psychology
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Dr. Eugene Charniak
Brown University
Computer Science Department
Providence, RI 02912

Dr. Paul R. Chatelier
OUSDRE
Pentagon
Washington, DC 20350-2000

Dr. Michelene Chi
Learning R & D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

Mr. Raymond E. Christal
AFHRL/MOE
Brooks AFB, TX 78235

Dr. Yee-Yeen Chu
Perceptronics, Inc.
21111 Erwin Street
Woodland Hills, CA 91367-371

Dr. William Clancey
Stanford University
Knowledge Systems Laboratory
701 Welch Road, Bldg. C
Palo Alto, CA 94304

Chief of Naval Education
and Training
Liaison Office
Air Force Human Resource Lab
Operations Training Division
Williams AFB, AZ 85224

Assistant Chief of Staff
for Research, Development,
Test, and Evaluation
Naval Education and
Training Command (N-5)
NAS Pensacola, FL 32508

Dr. Allan M. Collins
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

Dr. John J. Collins
Director, Field Research
Office, Orlando
NPRDC Liaison Officer
NTSC Orlando, FL 32813

Dr. Stanley Collyer
Office of Naval Technology
Code 222
800 N. Quincy Street
Arlington, VA 22217-5000

Dr. Lynn A. Cooper
Learning R&D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

Dr. Meredith P. Crawford
American Psychological Association
Office of Educational Affairs
1200 17th Street, N.W.
Washington, DC 20036

Dr. Hans Crombag
University of Leyden
Education Research Center
Boerhaavelaan 2
2334 EN Leyden
The NETHERLANDS

LT Judy Crookshanks
Chief of Naval Operations
OP-112G5
Washington, DC 20370-2000

Dr. Kenneth B. Cross
Anacapa Sciences, Inc.
P.O. Drawer Q
Santa Barbara, CA 93102

Dr. Mary Cross
Department of Education
Adult Literacy Initiative
Room 4145
400 Maryland Avenue, SW
Washington, DC 20202

CTB/McGraw-Hill Library
2500 Garden Road
Monterey, CA 93940

CAPT P. Michael Curran
Office of Naval Research
800 N. Quincy St.
Code 125
Arlington, VA 22217-5000

Dr. Cary Czichon
Intelligent Instructional Sy
Texas Instruments AI Lab
P.O. Box 660245
Dallas, TX 75266

Bryan Dallman
AFHRL/LRT
Lowry AFB, CO 80230

LT John Deaton
ONR Code 125
800 N. Quincy Street
Arlington, VA 22217-5000

Dr. Natalie Dehn
Department of Computer and
Information Science
University of Oregon
Eugene, OR 97403

Dr. Gerald F. DeJong
Artificial Intelligence Grou
Coordinated Science Laborato
University of Illinois
Urbana, IL 61801

Goery Delacote
Directeur de L'informatique
Scientifique et Technique
CNRS
15, Quai Anatole France
75700 Paris FRANCE

Mr. Robert Denton
AFMPC/MPCYPR
Randolph AFB, TX 78150

Mr. Paul DiRenzo
Commandant of the Marine Corps
Code LBC-4
Washington, DC 20380

Dr. R. K. Dismukes
Associate Director for Life Sciences
AFOSR
Bolling AFB
Washington, DC 20332

Defense Technical
Information Center
Cameron Station, Bldg 5
Alexandria, VA 22314
Attn: TC
(12 Copies)

Dr. Thomas M. Duffy
Communications Design Center
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Barbara Eason
Military Educator's
Resource Network
InterAmerica Research Associates
1555 Wilson Blvd
Arlington, VA 22209

Edward E. Eddowes
CNATRA N301
Naval Air Station
Corpus Christi, TX 78419

Dr. John Ellis
Navy Personnel R&D Center
San Diego, CA 92252

Dr. Richard Elster
Deputy Assistant Secretary
of the Navy (Manpower)
OASN (M&RA)
Department of the Navy
Washington, DC 20350-1000

Dr. Susan Embretson
University of Kansas
Psychology Department
426 Fraser
Lawrence, KS 66045

Dr. Randy Engle
Department of Psychology
University of South Carolina
Columbia, SC 29208

Lt. Col Rich Entlich
HQ, Department of the Army
OCSA(DACS-DPM)
Washington, DC 20310

Dr. Susan Epstein
Hunter College
144 S. Mountain Avenue
Montclair, NJ 07042

Dr. William Epstein
University of Wisconsin
W. J. Brogden Psychology Bld
1202 W. Johnson Street
Madison, WI 53706

ERIC Facility-Acquisitions
4833 Rugby Avenue
Bethesda, MD 20014

Dr. K. Anders Ericsson
University of Colorado
Department of Psychology
Boulder, CO 80309

Dr. Edward Esty
Department of Education, OER
Room 717D
1200 19th St., NW
Washington, DC 20208

Dr. Beatrice J. Farr
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Marshall J. Farr
2520 North Vernon Street
Arlington, VA 22207

Dr. Pat Federico
Code 511
NPRDC
San Diego, CA 92152-6800

Dr. Jerome A. Feldman
University of Rochester
Computer Science Department
Rochester, NY 14627

Dr. Paul Feltovich
Southern Illinois University
School of Medicine
Medical Education Department
P.O. Box 3926
Springfield, IL 62708

Mr. Wallace Feurzeig
Educational Technology
Bolt Beranek & Newman
10 Moulton St.
Cambridge, MA 02238

Dr. Craig I. Fields
ARPA
1400 Wilson Blvd.
Arlington, VA 22209

Dr. Gerhard Fischer
University of Colorado
Department of Computer Science
Boulder, CO 80309

J. D. Fletcher
9931 Corsica Street
Vienna VA 22180

Dr. Linda Flower
Carnegie-Mellon University
Department of English
Pittsburgh, PA 15213

Dr. Kenneth D. Forbus
University of Illinois
Department of Computer Science
1304 West Springfield Avenue
Urbana, IL 61801

Dr. Barbara A. Fox
University of Colorado
Department of Linguistics
Boulder, CO 80309

Dr. Carl H. Frederiksen
McGill University
3700 McTavish Street
Montreal, Quebec H3A 1Y2
CANADA

Dr. John R. Frederiksen
Bolt Beranek & Newman
50 Moulton Street
Cambridge, MA 02138

Dr. Norman Frederiksen
Educational Testing Service
Princeton, NJ 08541

Dr. Alfred R. Fregly
AFOSR/NL
Bolling AFB, DC 20332

Dr. Bob Frey
Commandant (G-P-1/2)
USCG HQ
Washington, DC 20593

Dr. Alinda Friedman
Department of Psychology
University of Alberta
Edmonton, Alberta
CANADA T6G 2E9

Julie A. Gadsden
Information Technology
Applications Division
Admiralty Research Establish
Portsdown, Portsmouth PO6 4A
UNITED KINGDOM

Dr. R. Edward Geiselman
Department of Psychology
University of California
Los Angeles, CA 90024

Dr. Michael Genesereth
Stanford University
Computer Science Department
Stanford, CA 94305

Dr. Dedre Gentner
University of Illinois
Department of Psychology
603 E. Daniel St.
Champaign, IL 61820

Dr. Robert Glaser
Learning Research
& Development Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15260

Dr. Arthur M. Glenberg
University of Wisconsin
W. J. Brogden Psychology Bldg.
1202 W. Johnson Street
Madison, WI 53706

Dr. Marvin D. Glock
13 Stone Hall
Cornell University
Ithaca, NY 14853

Dr. Gene L. Gloye
Office of Naval Research
Detachment
1030 E. Green Street
Pasadena, CA 91106-2485

Dr. Sam Glucksberg
Department of Psychology
Princeton University
Princeton, NJ 08540

Dr. Joseph Goguen
Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Dr. Sherrie Gott
AFHRL/MODJ
Brooks AFB, TX 78235

Jordan Grafman, Ph.D.
2021 Lyttonsville Road
Silver Spring, MD 20910

Dr. Richard H. Granger
Department of Computer Science
University of California, Irvine
Irvine, CA 92717

Dr. Wayne Gray
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. James G. Greeno
University of California
Berkeley, CA 94720

H. William Greenup
Education Advisor (E031)
Education Center, MCDEC
Quantico, VA 22134

Dipl. Pad. Michael W. Habon
Universitat Dusseldorf
Erziehungswissenschaftliches
Universitatsstr. 1
D-4000 Dusseldorf 1
WEST GERMANY

Prof. Edward Haertel
School of Education
Stanford University
Stanford, CA 94305

Dr. Henry M. Halff
Halff Resources, Inc.
4918 33rd Road, North
Arlington, VA 22207

Dr. Ronald K. Hambleton
Prof. of Education & Psychol
University of Massachusetts
at Amherst
Hills House
Amherst, MA 01003

Dr. Cheryl Hamel
NTSC
Orlando, FL 32813

Dr. Bruce W. Hamill
Johns Hopkins University
Applied Physics Laboratory
Johns Hopkins Road
Laurel, MD 20707

Dr. Ray Hannapel
Scientific and Engineering
Personnel and Education
National Science Foundation
Washington, DC 20550

Stevan Harnad
Editor, The Behavioral and
Brain Sciences
20 Nassau Street, Suite 240
Princeton, NJ 08540

Mr. William Hartung
PEAM Product Manager
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Wayne Harvey
SRI International
333 Ravenswood Ave.
Room B-S324
Menlo Park, CA 94025

Dr. Reid Hastie
Northwestern University
Department of Psychology
Evanston, IL 60201

Prof. John R. Hayes
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

Dr. Barbara Hayes-Roth
Department of Computer Science
Stanford University
Stanford, CA 95305

Dr. Frederick Hayes-Roth
Teknowledge
525 University Ave.
Palo Alto, CA 94301

Dr. Joan I. Heller
505 Haddon Road
Oakland, CA 94606

Dr. James Hendler
Dept. of Computer Science
University of Maryland
College Park, MD 20742

Dr. Jim Hollan
Intelligent Systems Group
Institute for
Cognitive Science (C-015)
UCSD
La Jolla, CA 92093

Dr. Melissa Holland
Army Research Institute for
Behavioral and Social Sci
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Robert W. Holt
Department of Psychology
George Mason University
4400 University Drive
Fairfax, VA 22030

Dr. Keith Holyoak
University of Michigan
Human Performance Center
330 Packard Road
Ann Arbor, MI 48109

Prof. Lutz F. Hornke
Institut fur Psychologie
RWTH Aachen
Jaegerstrasse 17/19
D-5100 Aachen
WEST GERMANY

Mr. Dick Hoshaw
OP-135
Arlington Annex
Room 2834
Washington, DC 20350

Dr. James Howard
Dept. of Psychology
Human Performance Laboratory
Catholic University of
America
Washington, DC 20064

Dr. Steven Hunka
Department of Education
University of Alberta
Edmonton, Alberta
CANADA

Dr. Earl Hunt
Department of Psychology
University of Washington
Seattle, WA 98105

Dr. Ed Hutchins
Intelligent Systems Group
Institute for
Cognitive Science (C-015)
UCSD
La Jolla, CA 92093

Dr. Dillon Inouye
WICAT Education Institute
Provo, UT 84057

Dr. Alice Isen
Department of Psychology
University of Maryland
Catonsville, MD 21228

Dr. Zachary Jacobson
Bureau of Management Consulting
365 Laurier Avenue West
Ottawa, Ontario K1A 0S5
CANADA

Dr. Robert Jannarone
Department of Psychology
University of South Carolina
Columbia, SC 29208

Dr. Claude Janvier
Directeur, CIRADE
Universite' du Quebec a Montreal
Montreal, Quebec H3C 3P8
CANADA

COL Dennis W. Jarvi
Commander
AFHRL
Brooks AFB, TX 78235-5601

Dr. Robin Jeffries
Hewlett-Packard Laboratories
P.O. Box 10490
Palo Alto, CA 94303-0971

Dr. Robert Jernigan
Decision Resource Systems
5595 Vantage Point Road
Columbia, MD 21044

Margaret Jerome
c/o Dr. Peter Chandler
83, The Drive
Hove
Sussex
UNITED KINGDOM

Dr. Joseph E. Johnson
Assistant Dean for
Graduate Studies
College of Science and Mathe
University of South Carolina
Columbia, SC 29208

Dr. Richard Johnson
Boise State University
Simplot/Micron
Technology Center
Boise, ID 83725

CDR Tom Jones
ONR Code 125
800 N. Quincy Street
Arlington, VA 22217-5000

Dr. Douglas A. Jones
P.O. Box 6640
Lawrenceville
NJ 08648

Col. Dominique Jouslin de No
Etat-Major de l'Armee de Ter
Centre de Relations Humaines
3 Avenue Octave Greard
75007 Paris
FRANCE

Dr. Marcel Just
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

Dr. Milton S. Katz
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Scott Kelso
Haskins Laboratories,
270 Crown Street
New Haven, CT 06510

Dr. Dennis Kibler
University of California
Department of Information
and Computer Science
Irvine, CA 92717

Dr. David Kieras
University of Michigan
Technical Communication
College of Engineering
1223 E. Engineering Building
Ann Arbor, MI 48109

Dr. Peter Kincaid
Training Analysis
& Evaluation Group
Department of the Navy
Orlando, FL 32813

Dr. Walter Kintsch
Department of Psychology
University of Colorado
Campus Box 345
Boulder, CO 80302

Dr. Paula Kirk
Oakridge Associated Universities
University Programs Division
P.O. Box 117
Oakridge, TN 37831-0117

Dr. David Klahr
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

Dr. Mazie Knerr
Program Manager
Training Research Division
HumRRO
1100 S. Washington
Alexandria, VA 22314

Dr. Janet L. Kolodner
Georgia Institute of Technology
School of Information
& Computer Science
Atlanta, GA 30332

Dr. Stephen Kosslyn
Harvard University
1236 William James Hall
33 Kirkland St.
Cambridge, MA 02138

Dr. Kenneth Kotovsky
Department of Psychology
Community College of
Allegheny County
800 Allegheny Avenue
Pittsburgh, PA 15233

Dr. David H. Krantz
2 Washington Square Village
Apt. # 15J
New York, NY 10012

Dr. Benjamin Kuipers
University of Texas at Austi
Department of Computer Scien
T.S. Painter Hall 3.28
Austin, Texas 78712

Dr. David R. Lambert
Naval Ocean Systems Center
Code 441T
271 Catalina Boulevard
San Diego, CA 92152-6800

Dr. Pat Langley
University of California
Department of Information
and Computer Science
Irvine, CA 92717

M. Diane Langston
Communications Design Cente
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Dr. Jill Larkin
Carnegie-Mellon University
Department of Psychology
Pittsburgh, PA 15213

Dr. Robert Lawler
Information Sciences, FRL
GTE Laboratories, Inc.
40 Sylvan Road
Waltham, MA 02254

Dr. Paul E. Lehner
PAR Technology Corp.
7926 Jones Branch Drive
Suite 170
McLean, VA 22102

Dr. Alan M. Lesgold
Learning R&D Center
University of Pittsburgh
Pittsburgh, PA 15260

Dr. Alan Leshner
Deputy Division Director
Behavioral and Neural Sciences
National Science Foundation
1800 G Street
Washington, DC 20550

Dr. Jim Levin
University of California
Laboratory for Comparative
Human Cognition
D003A
La Jolla, CA 92093

Dr. John Levine
Learning R&D Center
University of Pittsburgh
Pittsburgh, PA 15260

Dr. Michael Levine
Educational Psychology
210 Education Bldg.
University of Illinois
Champaign, IL 61801

Dr. Charles Lewis
Faculteit Sociale Wetenschappen
Rijksuniversiteit Groningen
Oude Boteringestraat 23
9712GC Groningen
The NETHERLANDS

Dr. Clayton Lewis
University of Colorado
Department of Computer Science
Campus Box 430
Boulder, CO 80309

Library
Naval War College
Newport, RI 02940

Library
Naval Training Systems Cente
Orlando, FL 32813

Dr. Charlotte Linde
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Dr. Marcia C. Linn
Lawrence Hall of Science
University of California
Berkeley, CA 94720

Dr. Robert Linn
College of Education
University of Illinois
Urbana, IL 61801

Dr. Don Lyon
P. O. Box 44
Higley, AZ 85236

Dr. Jane Malin
Mail Code SR 111
NASA Johnson Space Center
Houston, TX 77058

Dr. William L. Maloy
Chief of Naval Education
and Training
Naval Air Station
Pensacola, FL 32508

Dr. Sandra P. Marshall
Dept. of Psychology
San Diego State University
San Diego, CA 92182

Dr. Manton M. Matthews
Department of Computer Scien
University of South Carolina
Columbia, SC 29208

Dr. Richard E. Mayer
Department of Psychology
University of California
Santa Barbara, CA 93106

Dr. James McBride
Psychological Corporation
c/o Harcourt, Brace,
Javanovich Inc.
1250 West 6th Street
San Diego, CA 92101

Dr. David J. McGuinness
Gallaudet College
800 Florida Avenue, N.E.
Washington, DC 20002

Dr. Kathleen McKeown
Columbia University
Department of Computer Science
New York, NY 10027

Dr. Joe McLachlan
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. James McMichael
Assistant for MPT Research,
Development, and Studies
OP 01B7
Washington, DC 20370

Dr. Barbara Means
Human Resources
Research Organization
1100 South Washington
Alexandria, VA 22314

Dr. Douglas L. Medin
Department of Psychology
University of Illinois
603 E. Daniel Street
Champaign, IL 61820

Dr. Arthur Melmed
U. S. Department of Education
724 Brown
Washington, DC 20208

Dr. Jose Mestre
Department of Physics
Hasbrouck Laboratory
University of Massachusetts
Amherst, MA 01003

Dr. Al Meyrowitz
Office of Naval Research
Code 1133
800 N. Quincy
Arlington, VA 22217-5000

Dr. Ryszard S. Michalski
University of Illinois
Department of Computer Scien
1304 West Springfield Avenue
Urbana, IL 61801

Prof. D. Michie
The Turing Institute
36 North Hanover Street
Glasgow G1 2AD, Scotland
UNITED KINGDOM

Dr. George A. Miller
Department of Psychology
Green Hall
Princeton University
Princeton, NJ 08540

Dr. James R. Miller
MCC
9430 Research Blvd.
Echelon Building #1, Suite 2
Austin, TX 78759

Dr. Mark Miller
Computer*Thought Corporation
1721 West Plano Parkway
Plano, TX 75075

Dr. Andrew R. Molnar
Scientific and Engineering
Personnel and Education
National Science Foundation
Washington, DC 20550

Dr. William Montague
NPRDC Code 13
San Diego, CA 92152-6800

Dr. Tom Moran
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

Dr. Melvyn Moy
Navy Personnel R & D Center
San Diego, CA 92152-6800

Dr. Allen Munro
Behavioral Technology
Laboratories - USC
1845 S. Elena Ave., 4th Floor
Redondo Beach, CA 90277

Director,
Decision Support
Systems Division, NMPC
Naval Military Personnel Command
N-164
Washington, DC 20370

Director,
Distribution Department, NMPC
N-4
Washington, DC 20370

Director,
Overseas Duty Support
Program, NMPC
N-62
Washington, DC 20370

Head, HRM Operations Branch,
NMPC
N-62F
Washington, DC 20370

Assistant for Evaluation,
Analysis, and MIS, NMPC
N-6C
Washington, DC 20370

Spec. Asst. for Research, Experimental & Academic Programs,
NTTC (Code 016)
NAS Memphis (75)
Millington, TN 38054

Director,
Research & Analysis Div.,
NAVCRUITCOM Code 22
4015 Wilson Blvd.
Arlington, VA 22203

Technical Director
Navy Health Research Center
P.O. Box 85122
San Diego, CA 92138

Leadership Management Educat
and Training Project Offi
Naval Medical Command
Code 05C
Washington, DC 20372

Technical Director,
Navy Health Research Ctr.
P.O. Box 85122
San Diego, CA 92138

Dr. T. Niblett
The Turing Institute
36 North Hanover Street
Glasgow G1 2AD, Scotland
UNITED KINGDOM

Dr. Richard E. Nisbett
University of Michigan
Institute for Social Researc
Room 5261
Ann Arbor, MI 48109

Dr. Mary Jo Nissen
University of Minnesota
N218 Elliott Hall
Minneapolis, MN 55455

Dr. Donald A. Norman
Institute for Cognitive Scie
University of California
La Jolla, CA 92093

Director, Training Laborato
NPRDC (Code 05)
San Diego, CA 92152-6800

Director, Manpower and Perso
Laboratory,
NPRDC (Code 06)
San Diego, CA 92152-6800

Director, Human Factors
& Organizational Systems
NPRDC (Code 07)
San Diego, CA 92152-6800

Fleet Support Office,
NPRDC (Code 301)
San Diego, CA 92152-6800

Library, NPRDC
Code P201L
San Diego, CA 92152-6800

Commanding Officer,
Naval Research Laboratory
Code 2627
Washington, DC 20390

Dr. Harold F. O'Neil, Jr.
School of Education - WPH 801
Department of Educational
Psychology & Technology
University of Southern California
Los Angeles, CA 90089-0031

Dr. Stellan Ohlsson
Learning R & D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

Director, Research Programs,
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5000

Office of Naval Research,
Code 1133
800 N. Quincy Street
Arlington, VA 22217-5000

Office of Naval Research,
Code 1142
800 N. Quincy St.
Arlington, VA 22217-5000

Office of Naval Research,
Code 1142EP
800 N. Quincy Street
Arlington, VA 22217-5000

Office of Naval Research,
Code 1142PT
800 N. Quincy Street
Arlington, VA 22217-5000
(6 Copies)

Director, Technology Programs,
Office of Naval Research
Code 12
800 North Quincy Street
Arlington, VA 22217-5000

Special Assistant for Marine
Corps Matters,
ONR Code 00MC
800 N. Quincy St.
Arlington, VA 22217-5000

Assistant for Long Range
Requirements,
CNO Executive Panel
OP 00K
2000 North Beauregard Street
Alexandria, VA 22311

Assistant for MPT Research,
Development and Studies
OP 01B7
Washington, DC 20370

Dr. Judith Orasanu
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Jesse Orlansky
Institute for Defense Analys
1801 N. Beauregard St.
Alexandria, VA 22311

Prof. Seymour Papert
20C-109
Massachusetts Institute
of Technology
Cambridge, MA 02139

CDR R. T. Parlette
Chief of Naval Operations
OP-112G
Washington, DC 20370-2000

Dr. James Paulson
Department of Psychology
Portland State University
P.O. Box 751
Portland, OR 97207

Dr. Douglas Pearse
DCIEM
Box 2000
Downsview, Ontario
CANADA

Dr. Virginia E. Pendergrass
Code 711
Naval Training Systems Center
Orlando, FL 32813-7100

Dr. Robert Penn
NPRDC
San Diego, CA 92152-6800

Dr. Nancy Pennington
University of Chicago
Graduate School of Business
1101 E. 58th St.
Chicago, IL 60637

Military Assistant for Training and
Personnel Technology,
OUSD (R & E)
Room 3D129, The Pentagon
Washington, DC 20301-3080

Dr. Ray Perez
ARI (PERI-II)
5001 Eisenhower Avenue
Alexandria, VA 2233

Dr. David N. Perkins
Educational Technology Center
337 Gutman Library
Appian Way
Cambridge, MA 02138

LCDR Frank C. Petho, MSC, USN
CNATRA Code N36, Bldg. 1
NAS
Corpus Christi, TX 78419

Administrative Sciences Department,
Naval Postgraduate S[illegible]ool
Monterey, CA 93940

Department of Computer Science,
Naval Postgraduate School
Monterey, CA 93940

Department of Operations Research,
Naval Postgraduate School
Monterey, CA 93940

Dr. Tjeerd Plomp
Twente University of Technol
Department of Education
P.O. Box 217
7500 AE ENSCHEDE

THE NETHERLANDS

Dr. Martha Polson
Department of Psychology
Campus Box 346
University of Colorado
Boulder, CO 80309

Dr. Peter Polson
University of Colorado
Department of Psychology
Boulder, CO 80309

Dr. Steven E. Poltrock
MCC
9430 Research Blvd.
Echelon Bldg #1
Austin, TX 78759-6509

Dr. Harry E. Pople
University of Pittsburgh
Decision Systems Laboratory
1360 Scaife Hall
Pittsburgh, PA 15261

Dr. Mary C. Potter
Department of Psychology
MIT (E-10-032)
Cambridge, MA 02139

Dr. Joseph Psotka
ATTN: PERI-1C
Army Research Institute
5001 Eisenhower Ave.
Alexandria, VA 22333

Dr. Roy Rada
National Library of Medicine
Bethesda, MD 20894

Dr. Lynne Reder
Department of Psychology
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Brian Reiser
Department of Psychology
Princeton University
Princeton, NJ 08544

Dr. James A. Reggia
University of Maryland
School of Medicine
Department of Neurology
22 South Greene Street
Baltimore, MD 21201

CDR Karen Reider
Naval School of Health Sciences
National Naval Medical Center
Bldg. 141
Washington, DC 20814

Dr. Fred Reif
Physics Department
University of California
Berkeley, CA 94720

Dr. Lauren Resnick
Learning R & D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

Dr. Mary S. Riley
Program in Cognitive Science
Center for Human Information
Processing
University of California
La Jolla, CA 92093

William Rizzo
Code 712
Naval Training Systems Center
Orlando, FL 32813

Dr. Linda G. Roberts
Science, Education, and
Transportation Program
Office of Technology Assessment
Congress of the United States
Washington, DC 20510

Dr. Andrew M. Rose
American Institutes
for Research
1055 Thomas Jefferson St., NW
Washington, DC 20007

Dr. William B. Rouse
Search Technology, Inc.
25-b Technology Park/Atlanta
Norcross, GA 30092

Dr. Donald Rubin
Statistics Department
Science Center, Room 608
1 Oxford Street
Harvard University
Cambridge, MA 02138

Dr. David Rumelhart
Center for Human
Information Processing
Univ. of California
La Jolla, CA 92093

Ms. Riitta Ruotsalainen
General Headquarters
Training Section
Military Psychology Office
PL 919
SF-00101 Helsinki 10, FINLAN

Dr. Michael J. Samet
Perceptronics, Inc
6271 Variel Avenue
Woodland Hills, CA 91364

Dr. Roger Schank
Yale University
Computer Science Department
P.O. Box 2158
New Haven, CT 06520

Dr. W. L. Scherlis
Dept. of Computer Science
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Mrs. Birgitte Schneidelbach
Forsvarets Center for Leders
Christianshavns Voldgade 8
1424 Kobenhavn K
DENMARK

Dr. Walter Schneider
Learning R&D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15260

Dr. Alan H. Schoenfeld
University of California
Department of Education
Berkeley, CA 94720

Dr. Janet Schofield
Learning R&D Center
University of Pittsburgh
Pittsburgh, PA 15260

Dr. Marc Sebrechts
Department of Psychology
Wesleyan University
Middletown, CT 06475

Dr. Judith Segal
OERI
555 New Jersey Ave., NW
Washington, DC 20208

Dr. Robert J. Seidel
US Army Research Institute
5001 Eisenhower Ave.
Alexandria, VA 22333

Dr. Ramsay W. Selden
Assessment Center
CCSSO
Suite 379
400 N. Capitol, NW
Washington, DC 20001

Dr. W. Steve Sellman
OASD(MRA&L)
2B269 The Pentagon
Washington, DC 20301

Dr. Sylvia A. S. Shafto
OERI
555 New Jersey Ave., NW
Washington, DC 20208

Dr. T. B. Sheridan
Dept. of Mechanical Engineering
MIT
Cambridge, MA 02139

Dr. Ben Shneiderman
Dept. of Computer Science
University of Maryland
College Park, MD 20742

Dr. Ted Shortliffe
Computer Science Department
Stanford University
Stanford, CA 94305

Dr. Lee Shulman
Stanford University
1040 Cathcart Way
Stanford, CA 94305

Dr. Randall Shumaker
Naval Research Laboratory
Code 7510
4555 Overlook Avenue, S.W.
Washington, DC 20375-5000

Dr. Robert S. Siegler
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

Dr. Herbert A. Simon
Department of Psychology
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

LTCOL Robert Simpson
Defense Advanced Research
Projects Administration
1400 Wilson Blvd.
Arlington, VA 22209

Dr. Zita M Simutis
Instructional Technology
Systems Area
ARI
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. H. Wallace Sinaiko
Manpower Research
and Advisory Services
Smithsonian Institution
801 North Pitt Street
Alexandria, VA 22314

Dr. Derek Sleeman
Stanford University
School of Education
Stanford, CA 94305

Dr. Charles F. Smith
North Carolina State Univers
Department of Statistics
Raleigh, NC 27695-8703

Dr. Edward E. Smith
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

Dr. Linda B. Smith
Department of Psychology
Indiana University
Bloomington, IN 47405

Dr. Alfred F. Smode
Senior Scientist
Code 07A
Naval Training Systems Center
Orlando, FL 32813

Dr. Richard E. Snow
Department of Psychology
Stanford University
Stanford, CA 94306

Dr. Elliot Soloway
Yale University
Computer Science Department
P.O. Box 2158
New Haven, CT 06520

Dr. Richard Sorensen
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Kathryn T. Spoehr
Brown University
Department of Psychology
Providence, RI 02912

James J. Staszewski
Research Associate
Carnegie-Mellon University
Department of Psychology
Schenley Park
Pittsburgh, PA 15213

Dr. Marian Stearns
SRI International
333 Ravenswood Ave.
Room B-S324
Menlo Park, CA 94025

Dr. Robert Sternberg
Department of Psychology
Yale University
Box 11A, Yale Station
New Haven, CT 06520

Dr. Albert Stevens
Bolt Beranek & Newman, Inc.
10 Moulton St.
Cambridge, MA 02238

Dr. Paul J. Sticha
Senior Staff Scientist
Training Research Division
HumRRO
1100 S. Washington
Alexandria, VA 22314

Dr. Thomas Sticht
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. David Stone
KAJ Software, Inc.
3420 East Shea Blvd.
Suite 161
Phoenix, AZ 85028

Cdr Michael Suman, PD 303
Naval Training Systems Cente
Code N51, Comptroller
Orlando, FL 32813

Dr. Hariharan Swaminathan
Laboratory of Psychometric a
Evaluation Research
School of Education
University of Massachusetts
Amherst, MA 01003

Mr. Brad Sympson
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. John Tangney
AFOSR/NL
Bolling AFB, DC 20332

Dr. Kikumi Tatsuoka
CERL
252 Engineering Research
Laboratory
Urbana, IL 61801

Dr. Martin M. Taylor
DCIEM
Box 2000
Downsview, Ontario
CANADA

Dr. Perry W. Thorndyke
FMC Corporation
Central Engineering Labs
1185 Coleman Avenue, Box 580
Santa Clara, CA 95052

Major Jack Thorpe
DARPA
1400 Wilson Blvd.
Arlington, VA 22209

Dr. Douglas Towne
Behavioral Technology Labs
1845 S. Elena Ave.
Redondo Beach, CA 90277

Dr. Amos Tversky
Stanford University
Dept. of Psychology
Stanford, CA 94305

Dr. James Tweeddale
Technical Director
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Paul Twohig
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Headquarters, U. S. Marine Corps
Code MPI-20
Washington, DC 20380

Dr. Kurt Van Lehn
Department of Psychology
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

Dr. Beth Warren
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

Dr. David J. Weiss
N660 Elliott Hall
University of Minnesota
75 E. River Road
Minneapolis, MN 55455

Roger Weissinger-Baylon
Department of Administrative
Sciences
Naval Postgraduate School
Monterey, CA 93940

Dr. Donald Weitzman
MITRE
1820 Dolley Madison Blvd.
MacLean, VA 22102

Dr. Keith T. Wescourt
FMC Corporation
Central Engineering Labs
1185 Coleman Ave., Box 580
Santa Clara, CA 95052

Dr. Douglas Wetzel
Code 12
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Barbara White
Bolt Beranek & Newman, Inc.
10 Moulton Street
Cambridge, MA 02238

LCDR Cory deGroot Whitehead
Chief of Naval Operations
OP-112G1
Washington, DC 20370-2000

Dr. Michael Williams
IntelliCorp
1975 El Camino Real West
Mountain View, CA 94040-2216

Dr. Hilda Wing
Army Research Institute
5001 Eisenhower Ave.
Alexandria, VA 22333

A. E. Winterbauer
Research Associate
Electronics Division
Denver Research Institute
University Park
Denver, CO 80208-0454

Dr. Robert A. Wisher
U.S. Army Institute for the
Behavioral and Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Martin F. Wiskoff
Navy Personnel R & D Center
San Diego, CA 92152-6800

Dr. Merlin C. Wittrock
Graduate School of Education
UCLA
Los Angeles, CA 90024

Mr. John H. Wolfe
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Wallace Wulfeck, III
Navy Personnel R&D Center
San Diego, CA 92152-6800

Dr. Joe Yasatuke
AFHRL/LRT
Lowry AFB, CO 80230

Dr. Masoud Yazdani
Dept. of Computer Science
University of Exeter
Exeter EX4 4QL
Devon, ENGLAND

Major Frank Yohannan, USMC
Headquarters, Marine Corps
(Code MPI-20)
Washington, DC 20380

Mr. Carl York
System Development Foundation
181 Lytton Avenue
Suite 210
Palo Alto, CA 94301

Dr. Joseph L. Young
Memory & Cognitive
Processes
National Science Foundation
Washington, DC 20550

Dr. Steven Zornetzer
Office of Naval Research
Code 1140
800 N. Quincy St.
Arlington, VA 22217-5000

Dr. Michael J. Zyda
Naval Postgraduate School
Code 52CK
Monterey, CA 93943-5100