THE ADVANTAGES OF ABSTRACT CONTROL KNOWLEDGE IN EXPERT SYSTEM DESIGN

William J. Clancey

Heuristic Programming Project
Computer Science Department
Stanford University
Stanford, CA 94305

ABSTRACT

A poorly designed knowledge base can be as cryptic as an arbitrary program and just as difficult to maintain. Representing control knowledge abstractly, separately from domain facts and relations, makes the design more transparent and explainable. A body of abstract control 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.*

I INTRODUCTION

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;
- 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.

A knowledge base is like a traditional program in that maintaining it requires having a good understanding of the underlying design. That is, you need to know how the parts of the knowledge base are expected to interact in problem solving. Depending on the representation, this includes knowing how default and judgmental knowledge interact, whether rule clauses can be reordered, when attached procedures are applied, how constraints are inherited and ordered, etc. One way to provide this understanding is to have the program explain its reasoning, using an internal description of its own design (Davis, 1976), (Swartout, 1977). However, problems encountered in understanding traditional programs--poorly-structured code, implicit side-effects, and inadequate documentation--carry over to knowledge-based programming and naturally limit the capabilities of explanation programs. For example, a knowledge base might arbitrarily combine reasoning strategies with facts about the domain. Implicit, procedurally-embedded knowledge cannot be articulated by an explanation system (Swartout, 1981), (Clancey, 1983) and is not visible to guide the program maintainer (see (Ennis, 1982) for an entertaining study of this problem).

This paper argues that *an important design principle for building expert systems is to represent all control knowledge abstractly, separate from the domain knowledge it operates upon.* This idea is illustrated with examples from the NEOMYCIN system (Clancey, 1981). There are many scientific, engineering, and practical benefits. The difficulty of attaining this ideal design is also considered.

II WHAT IS ABSTRACT CONTROL KNOWLEDGE?

"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.

*This research has been supported in part by ONR and ARI Contract N00014-79C-0302. Computational resources have been provided by the SUMEX-AIM facility (NIH grant RR00785).

A. 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) 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 (Clancey, 1981) and (Clancey and Bock, 1982) and are not important here). 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.

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

B. 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.

III STUDYING ABSTRACT STRATEGIES AND STRUCTURAL RELATIONS

In NEOMYCIN, domain findings and disorders are related in the way shown above, and there are approximately 75 metarules that constitute a procedure for doing diagnosis. Besides abstract domain relations, such as SUBSUMES, NEOMYCIN's metarules reference:

- Knowledge about metarules and tasks: (static) the argument of a task, whether metarules are to be applied iteratively, 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: the active hypotheses, whether a hypothesis was pursued, cumulative belief for a hypothesis, rules using a finding that are "in focus", a strong competitor to a given hypothesis, etc.

These concepts form the vocabulary for a model of diagnosis, the terms in which expert behavior is interpreted and strategies are expressed.

An unexpected effect 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. There are two specific products: *a body of abstract control knowledge* that can itself be studied, as well as applied in other problem domains, and *a language for representing knowledge about disorders* (in terms of causality, subtype, etc.). We call these abstract relations *structural relations.*

Structural 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, structural 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 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 structural 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 structural 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 structural 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.

IV GIVEN THE BENEFITS, CAN IT BE DONE?

An initial reaction might be that for some domains there are no patterns for using knowledge--no abstract strategies--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 spacial 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 NEOMYCIN's tasks.

Importantly, the *rationale for the abstract plan* itself is not explicit in any of these programs. For example, NEOMYCIN'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.

Even within the practical bounds of what we make explicit, it might be argued that representing procedures abstractly is much more difficult than stating individual situation-specific rules. This might differ from person to person; certainly in medicine some physicians are better than others at stating how they reason abstractly. A good heuristic might be to work with good teachers, for they are most likely to have extracted the principles so they can be taught to students.

There is certainly an initial cost 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. 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.

The cost aside, it appears that there is no way to get strategic explanations without making domain relations explicit and stating strategies separately. This was the conclusion of Swartout, who was led to conclude 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.

NEOMYCIN's factoring of knowledge into domain and strategic knowledge bases is comparable to the input requirements of Swartout's automatic programming system. However, NEOMYCIN interprets its domain knowledge, rather than instantiating its abstract strategies in a compiled program. (Maintaining the separation is important so the metarules can be used in student modeling (London and Clancey, 1982).) Moreover, NEOMYCIN's strategies are abstract, unlike the domain-specific "principles" used in Swartout's program. This design decision was originally motivated by our desire to replicate the kind of explanations given by teachers (Hasling, 1983). However, we now realize that representing control knowledge abstractly has engineering and scientific benefits as well.

V ADVANTAGES OF THE APPROACH

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.
  - 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.

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). Constructing a knowledge base becomes a matter of declaring knowledge relations. HOW the knowledge will be used needn't be simultaneously and redundantly specified.

An analogy can be made with GUIDON (Clancey, 1979) (Clancey, 1982), whose body of abstract teaching rules make 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.

Another analogy can be made with 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 expert systems.

References

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

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 IJCAI*, pages 829-836, 1981. (Revised version to appear in Clancey and Shortliffe (editors), *Readings in medical artificial intelligence: The first decade*, Addison-Wesley, 1983).

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. (Revised version to appear in the *Journal of Computer-Based Instruction*, 1983).

Clancey, W. J. The epistemology of a rule-based expert system: A framework for explanation. *Artificial Intelligence*, 1983, *20(3)*, 215-251. (Also to appear in Buchanan and Shortliffe (editors), *Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, 1983).

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

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.

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

Ennis, S. P. *Expert systems: A user's perspective of some current tools*, in *Proceedings of the Second AAAI*, pages 319-321, August, 1982.

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

Friedland, P. *Knowledge-based experiment design in molecular genetics*, in *Proceedings of the Sixth IJCAI*, pages 285-287, 1979.

Hasling, D. W. Abstract explanations of strategy in a diagnostic consultation system. (To appear in the *Proceedings of AAAI-83*).

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.

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.

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

Stefik, M. J. *Planning with constraints*. PhD thesis, Computer Science Department, Stanford University, 1980.

Swartout, W. R. *A digitalis therapy advisor with explanations*. Technical report 176, MIT Laboratory for Computer Science, February 1977.

Swartout W. R. *Explaining and justifying in expert consulting programs*, in *Proceedings of the Seventh IJCAI*, August, 1981. (Also to appear in Clancey and Shortliffe (editors), *Readings in medical artificial intelligence: The first decade*, Addison-Wesley, 1983).