Knowledge Systems Laboratory April 1986

Working Paper No. 86-34

Guidon-Debug: The student as knowledge engineer

William J. Clancey, Mark Richer, David C. Wilkins, Steve Barnhouse,
Curt Kapsner, David Leserman, John Macias, Arif Merchant, and Naomi Rodolitz

Stanford Knowledge Systems Laboratory
701 Welch Road, Building C
Palo Alto, CA 94304

Abstract

GUIDON-DEBUG is an instructional program in which a student learns by debugging a knowledge base. We attempt to teach specific medical facts, as well as general principles for critiquing and improving diagnostic reasoning. The knowledge base to be debugged is provided by NEOMYCIN. The student communicates with GUIDON-DEBUG by editing the ordered record of questions asked by NEOMYCIN during a consultation, and by editing the patient-specific model that explains abnormal findings. GUIDON-DEBUG evaluates changes for consistency with NEOMYCIN's idealized diagnostic procedure and assists the student in generating and testing domain model revisions. Thus, the student takes on the role of knowledge engineer, with the aid of powerful tools for inspecting and modifying the diagnostic process. In contrast with other intelligent tutoring systems, every effort has been made to expose "the machine behind the curtain," enabling the student to become engaged in our modeling methodology, rather than imposing upon him a predetermined view of the world.

1. Introduction

Since the 1950s, the idea of using the computer as a communication medium has inspired the development of instructional programs (Sleeman and Brown, 1982, Clancey, 1982a, Wenger 86, 1986). Besides the obvious approach of selectively presenting text and problems to a student, computer simulations are widely used, allowing the student to explore how some physical system works (Hollan, et al., 1984). Another approach has been to teach programming itself, with the view that exploring computational models in geometry, physics, music, etc. teaches general problem-solving or metacognitive skills (Papert, 1980, Goldberg, 1979). Although several AI-based instructional programs are designed as "omniscient" tutors that strongly guide interaction with the student (Stevens and Collins, 1977, Clancey, 1982b, Reiser, et al., 1985), the trend is towards balancing guidance and feedback with open-ended exploration (Brown, 1976, diSessa, 1984).

The GUIDON-2 family of instructional programs integrates different instructional approaches, including knowledge-based simulation, tutoring, and exploratory programming. The foundation for these programs is the NEOMYCIN consultation system (Clancey and Letsinger, 1984). Two programs in this series have been implemented: (1) GUIDON-WATCH (Richer and Clancey, 1985), a graphic interface that allows a student to *watch* and *inspect* diagnostic reasoning, and (2) GUIDON-DEBUG, the program described in this paper. Within the context of GUIDON-DEBUG, the student uses GUIDON-WATCH as a debugging tool to inspect the program's diagnostic reasoning, playing the role of knowledge engineer.

Part of the controversy concerning how to effectively use a computer for instruction derives from the limitations of knowledge bases: (1) They often *implicitly* represent knowledge we want to teach (Clancey, 1983a, Swartout, 1981); (2) they are often not based on a model of how humans organize knowledge and solve problems (Goldberg, 1973); (3) they generally have a fixed, single model of the world which may differ from the student's model (Stevens and Collins, 1978). We address these issues in our research.

In contrast to MYCIN (Shortliffe, 1976), NEOMYCIN is based on a model of how physicians diagnose patients (Clancey, 1984). Its diagnostic procedure is represented *explicitly*, separate from medical facts and heuristics. In general, the NEOMYCIN knowledge base significantly reconfigures and augments the original MYCIN program: Causal and subtype relations among patient data, pathophysiologic states, and diseases are explicitly represented. The domain is broadened to include diseases that might be confused with meningitis, such as subarachnoid hemorrhage, migraine, and tension headache, providing an opportunity for teaching differential

diagnosis.

NEOMYCIN's clean separation between medical facts, procedure, and their encoding is exploited in GUIDON-WATCH by providing interactive graphic displays. The interface enables students to browse and edit the knowledge base and view reasoning processes at any point in a consultation. Thus, to a significant degree we have realized the conception of a "glass-box" expert (Goldstein and Papert, 1977), finally made possible by advances in modeling, representing, and displaying expertise.

The single, fixed model of the world encoded in knowledge bases is a particularly important limitation for instruction. A student is not a blank slate to which we can copy over or transfer expertise from a program. The predominant view is that effective teaching benefits from—and may require—understanding the student's model of the world and *relating it* to the teacher's view. Thus, research in the past decade has studied the alternative models of the world that people have (Gentner and Stevens, 1983, Rouse and Morris, 1985) and how a program might construct a model of a student's understanding (Clancey, 1986a).

Viewing the student's learning task in terms of model-building, we provide instructional tools for building, reifying, and debugging models. The ODYSSEUS modeler (Wilkins, et al., 1986) described in this paper helps the student compare his understanding of the diagnostic process to NEOMYCIN's. Thus, the student can improve the knowledge base we provide and extend it according to his own understanding and interests. GUIDON-DEBUG, by providing guidance and feedback in a student-centered task, avoids the problems of a "lock-step" tutor, which must correctly understand the student at each moment.

Many research themes come together in this approach to teaching: Apprentice learning by debugging (Mitchell, et al., 1985); exploiting the symmetry between student modeling and knowledge acquisition (Barr, 1979a); emphasizing the reasoning process instead of its product (Dewey, 1964, Brown et al., 1977, Schoenfeld, 1981); reifying the reasoning process by graphics techniques (Brown, 1983, Hollan, et al., 1984, Richer and Clancey, 1985); and thinking about diagnosis and knowledge bases as models (Patil, 1981, Clancey, 1986b). These are considered briefly in Section 4.

2. Scenario

GUIDON-DEBUG is intended to be used by medical students who are beginning clincal training or clerkships. The patient cases presented to students are taken directly from actual patient charts and are entered in a patient data file.

An interaction with GUIDON-DEBUG has the following form:

- Run NEOMYCIN:
  - A patient case along with a set of knowledge base changes is selected;
  - NEOMYCIN runs the case in "quiet mode," i.e., nothing is displayed until the final diagnosis is presented;
- Browse:
  - The student is given the typescript of question and answer pairs in a scrollable window (Figure 2-1);
  - Using GUIDON-WATCH, the student inspects the program's reasoning and beliefs via menus and selectable items on the screen;
- Annotate and Edit:
  - The student can annotate NEOMYCIN's typescript to indicate questions to add, delete, or reorder (Figure 2-1);
  - To indicate additional changes to the knowledge base, the student edits NEOMYCIN's patient-specific model graph (Figure 2-3);
- Evaluate:
  - The ODYSSEUS modeling system evaluates the student's annotations for consistency with the diagnostic procedure (Figure 2-2);
  - The student re-runs the case to examine the effect of his changes.

Details are provided in the sections below.

2.1. Bug curriculum

The first step in using GUIDON-DEBUG is to run a NEOMYCIN consultation using a 'buggy' knowledge base. Each case is associated with one or more modifications to the knowledge base that will manifest in an *observable bug* in either the program's typescript of questions and answers or as a missing or incorrect link in the patient-specific model graph. The bugs are always missing, additional, or modified domain rules or relations. It is much more difficult (and unusual) to modify the diagnostic procedure, so this is not currently considered.

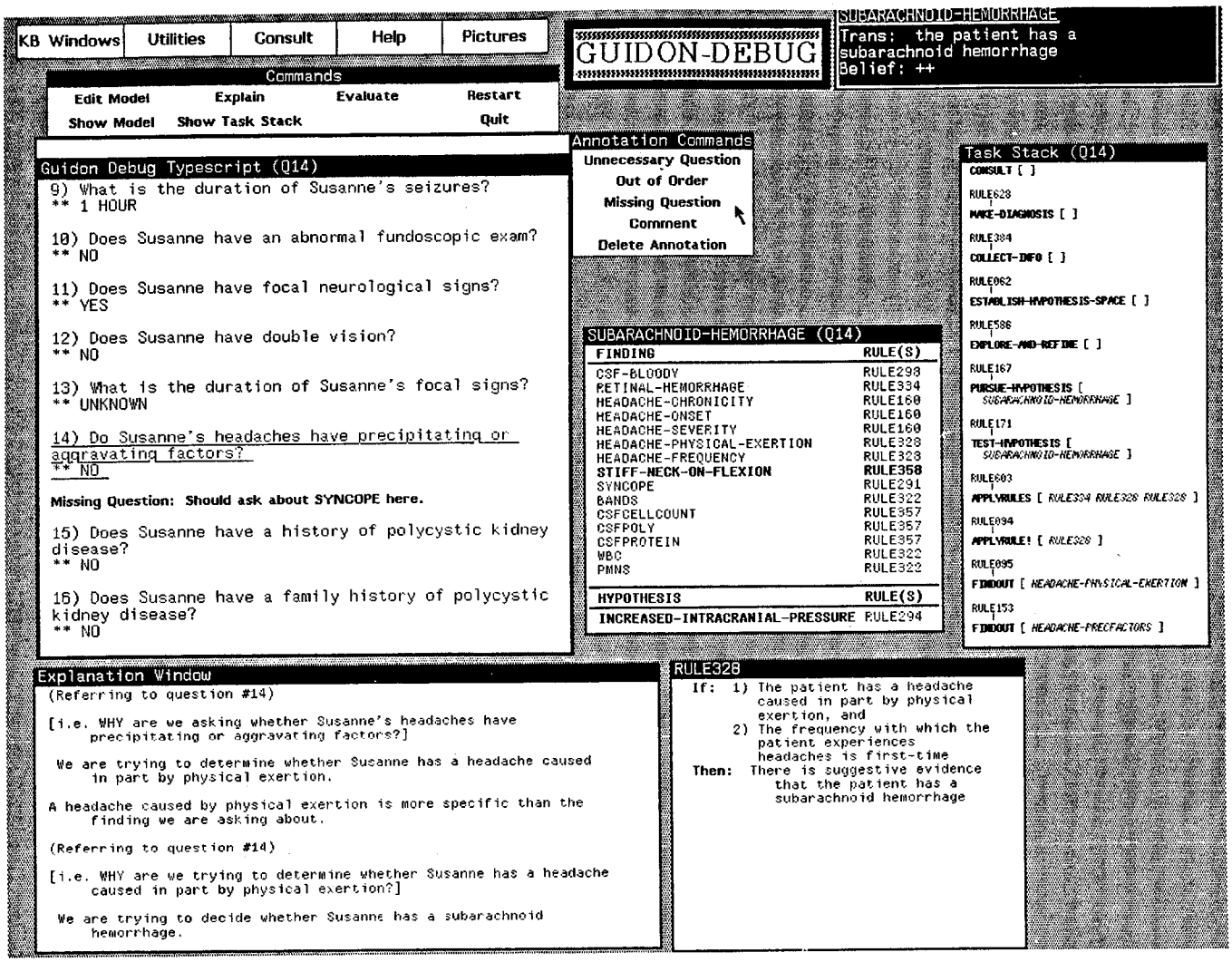

Figure 2-1: General layout of GUIDON-DEBUG

2.2. Browsing

The student can use the GUIDON-WATCH interface to browse the knowledge base and examine the program's reasoning history. Specifically, the student can browse a disease taxonomy, causal networks, evidence relations, and a history of the diagnostic strategy. Students can also "roll back" the process by selecting a question number (by buttoning the typescript) and displaying windows, which are revised to show their state at that time.

NEOMYCIN's explanation system is also an important tool for the student because it condenses lines of reasoning, omitting many of the details the student can see in the various windows (Figure 2-1) (Hasling, 1984). "WHY" explanations indicate only changes in focus (e.g., considering a new hypothesis) and skip over tasks focusing on domain rules.

2.3. Annotation

The way in which a student annotates the typescript is illustrated in Figure 2-1. In this example, the student has selected a question, examined the state of the consultation as it appeared at the time that question was asked, and indicated that the program should have asked about syncope at that point. In having a student annotate a typescript, we are giving him a tool for organizing his analysis of the buggy knowledge base: What questions seem out of place or are missing? What is the program trying to do? What domain facts is it using? Secondarily, the student may discover correct portions of the typescript that he does not understand. The same tools will help him learn about the program's model.

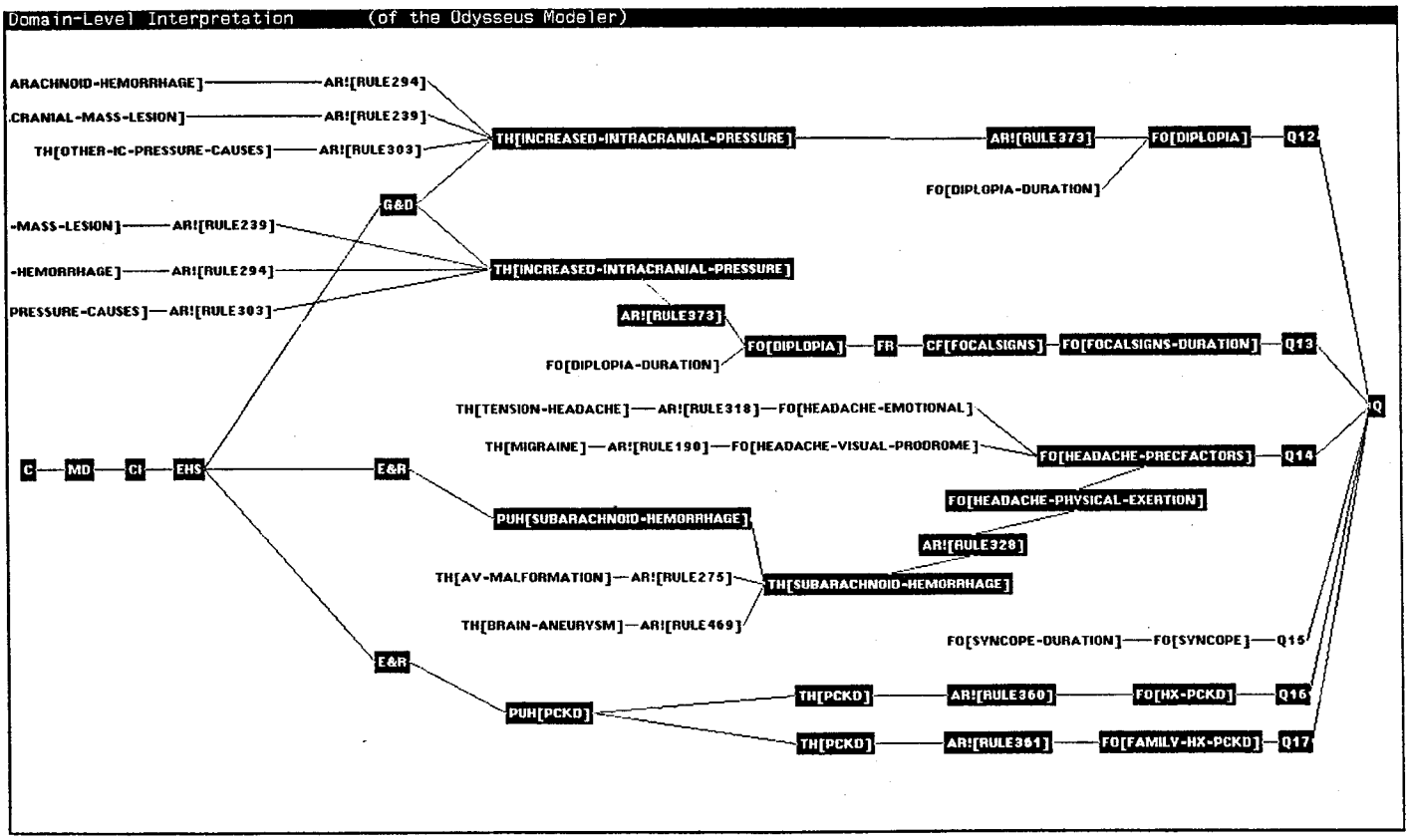

Figure 2-2: ODYSSEUS' parse of a student's edited typescript

2.4. Interpretation

After a student adds, deletes and reorders the questions of a NEOMYCIN consultation using the annotation facility, the modified typescript is critiqued by the ODYSSEUS modeling program. Figure 2-2 reflects the insertion of syncope after question 14. The inverted nodes indicates ODYSSEUS' parse of questions 12-17 using NEOMYCIN's procedure. Node "Q15" is not inverted because OYSSEUS could not find a consistent interpretation for the syncope question. Further dialogue with the student could determine that the student believes that subarachnoid hemorrhage causes syncope.

For the purpose of teaching, we can view the annotated typescript as showing what the student would have done, if he had solved the problem himself. The program attempts to *parse* the sequence of questions, that is, to determine if they are consistent with NEOMYCIN's diagnostic strategy. A gap in the parse suggests an underlying strategic or domain difference between the student and NEOMYCIN, resolved by asking the student to explain his reasoning and comparing it to the program (see Section 3).

2.5. Editing the PSM

The *patient-specific model* (PSM) reveals how NEOMYCIN's hypotheses "explain" the abnormal patient findings (Patil, 1981, Clancey, 1986b). As shown in Figure 2-3, the most specific hypotheses or diagnoses are displayed at the top of the graph and the patient findings at the bottom. A window below the PSM lists the findings that are not yet explained by the program (not shown in figure).

The user can edit the PSM to create or modify causal and subtype relations in the knowledge base. Editing is accomplished by direct manipulation of the PSM using the mouse. The user can (1) add a link between a disorder hypothesis and one or more abnormal findings or another hypothesis, (2) add an hypothesis or an abnormal finding, or (3) delete a link between any pair of nodes. The graph editor will automatically generate, delete, or modify rules in the knowledge base. As a convenience, the name of a finding or hypothesis may be buttoned and copied from another open window, such as the disease taxonomy or the unexplained findings window. Although other systems allow users to graphically edit a knowledge base, this represents the first system where a student debugs a knowledge base by graphically editing a dynamic domain model, that is, the PSM.

In Figure 2-3, a link is added between acute bacterial meningitis and seizures; a rule is added to the knowledge base, and the user is prompted for his degree of belief in this rule. In the

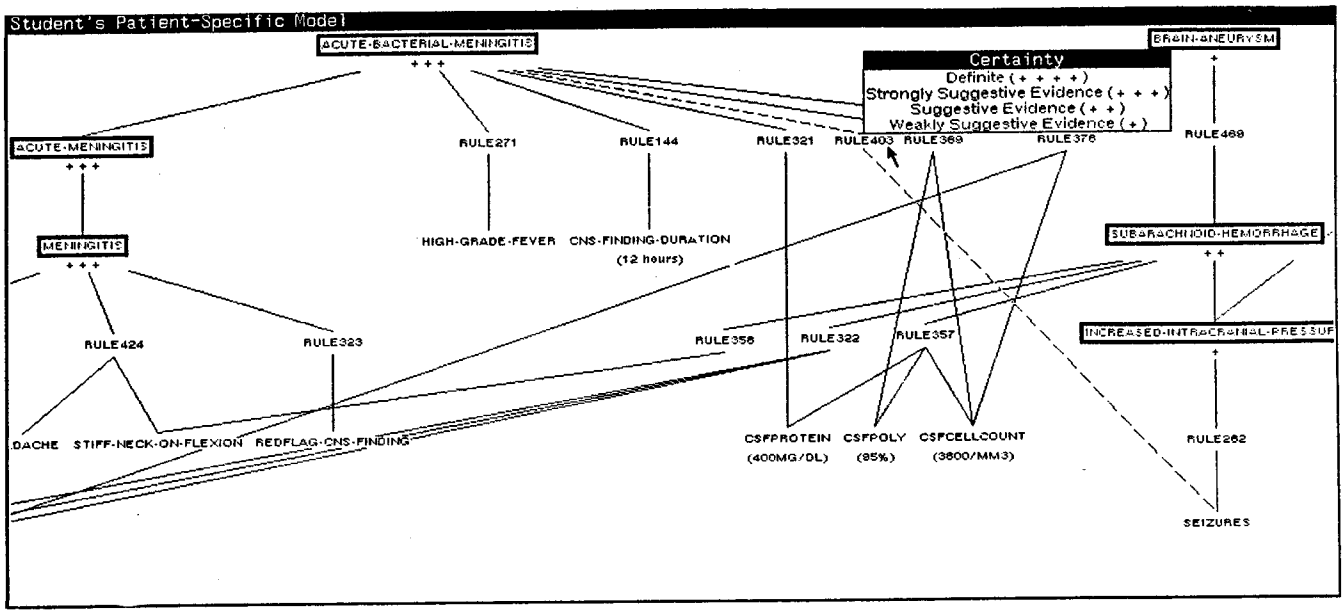

Figure 2-3: Editing NEOMYCIN's patient-specific model

current system, changes made via the PSM are not evaluated by GUIDON-DEBUG.

3. Implementation: How this is possible

Here we briefly describe the representation and analysis ideas that make GUIDON-DEBUG's display and interpretation features possible.

3.1. Neomycin: Abstract diagnostic strategy

To understand the advantages of NEOMYCIN, consider why a similar instructional program cannot be constructed from MYCIN:

- MYCIN does not represent causal and subtype relationships among diseases and patient findings explicitly. Therefore, its diagnosis cannot be shown as a support network in terminology familiar to the medical student.

- MYCIN's diagnostic strategy is implicit in its parameters and rules (Clancey, 1983a). Therefore, the disease model cannot be edited independently of how it is used by the program. Furthermore, the abstract, *functional model of strategy* in NEOMYCIN (Clancey, 1983b, Clancey, 1986a), enables ODYSSEUS to infer new medical relations by examining the strategic context in which questions are asked. This same approach is used in the MACSYMA ADVISOR (Genesereth, 1982a).

- MYCIN's implicit strategy omits most of the diagnostic strategies used by people, such as "triggering a hypothesis," "grouping hypotheses" and "asking a follow-up question" (Clancey, 1984). Therefore, a tutor based on MYCIN can neither recognize nor articulate these strategies.

In summary, these considerations make it possible for a person to understand NEOMYCIN and make it possible for us to construct programs that can understand NEOMYCIN (for display, explanation, and modeling).

Because GUIDON-DEBUG depends only on the language of the diagnostic procedure, it can be used for any knowledge base encoded in this representation (called HERACLES, for "heuristic classification shell"). In particular, GUIDON-DEBUG can be used with CASTER (Thompson and Clancey, 1986), a diagnostic program for sandcasting that uses the same procedure as NEOMYCIN.

3.2. Odysseus

In evaluating a student's annotated typescript, ODYSSEUS must be able to recognize that a certain question is strategically equivalent to what NEOMYCIN did. For example, in testing a particular hypothesis, there may be several findings that could be requested in any order. Reordering these questions will in turn move entire *subsequences* of questions, resulting from follow-ups and forward reasoning that triggered other lines of inquiry. Thus, it is not just a matter of reordering questions, but hierarchially reparsing the entire sequence.

The ODYSSEUS modeling cycle can be summarized as follows. For each question in sequence, ODYSSEUS:

1. Generates a set of interpretations of the action, that is, a bottom-up analysis of all possible reasons for asking the question;

2. Tests interpretations for plausibility (consistency with beliefs about the problem

solver's domain knowledge and existence of top-down strategy parse), and

3. If no viable interpretation remains, searches for a change of goal (diagnostic task and/or focus), flawed interpretation, or missing or discrepant domain knowledge.

For example, assume that the student has indicated that question 14 should be, "Has the patient experienced syncope?" The program notes that the preceding task is test-hypothesis with a focus of subarachnoid hemorrhage. Since no rule relating syncope and subarachnoid hemorrhage is in the knowledge base, ODYSSEUS searches for a possible change of focus. None leads to a line of reasoning that is consistent with the context and the diagnostic procedure.

If there is a clear surrounding context, as in the example (Figure 2-2), the program then asks the student if he is considering a relation between the established focus and his question. That is, does he believe that syncope causes or is caused by subarachnoid hemorrhage?

If this fails, the program considers the bottom-up parse, which is based on domain relations known to NEOMYCIN. If the student acknowledges that he has one of these interpretations in mind, the program explains why the diagnostic procedure does not generate that line of reasoning. Otherwise, the program asks the student what domain relation he has in mind, and determines whether adding this to the knowledge base results in a consistent parse. If not, there is a strategic difference as well.

4. Implications

Here we go beyond the operation of GUIDON-DEBUG to discuss the theory behind its design and relation to other research.

4.1. Instructional programs

The development of GUIDON-WATCH was influenced by Brown's ideas about "reifying the process," emphasizing the use of graphics for making the reasoning process concrete (Brown, 1983). We view a diagnosis as an object (the PSM) that is transformed over time by operators (NEOMYCIN's tasks), analogous to operators for transforming an algebraic equation. The PSM serves as an argument or proof showing the support for alternative hypotheses. We were influenced by Anderson's use of a graph for displaying the proof of a theorem (Anderson, et al., 1985), which shows a complete or partial solution as the theorem to be proved supported by axioms, other theorems and given information. With this kind of graph, different opportunities for completing or debugging a solution become visible. Together this research

demonstrates that making a partial solution concrete—reifying the *product*—is crucial for understanding the process in domains that lack a written notation for a partial solution. Without such a written notation in medical diagnosis, a teacher might not question how certain facts were explained, especially if the solution (e.g., a disease name) is correct, thus failing to detect bugs. Thus, GUIDON-DEBUG provides a notation that facilitates instruction, providing a new form of communication.

4.2. Critiquing model for knowledge acquisition

Tutorial programs typically have the student solve a problem and critique his solution. The *critiquing model* of consultation programs is similar: The user supplies his own answers and the program indicates important differences from what it would do (Clancey, 1978, Miller, et. al., 1983, Langlotz andd Shortliffe, 1983). This form of interaction keeps the user engaged in solving the problem, can be instructive, and places responsibility for the final decision with the user. GUIDON-DEBUG extends the critiquing model, by allowing the user to edit the typescript. Thus, the *process* by which the problem is solved can be criticized as well as the final *product.*

GUIDON-DEBUG builds on the approach used in TIERESIAS of debugging a knowledge base, providing additional tools for examining a program's reasoning. The windows separate out what would be a continous tracing requiring dozens of pages into different perspectives for understanding the reasoning process. The PSM in particular shows the support structure for the solution that TEIRESIAS could only implicitly unravel through a tedious rule and goal debugging trail. A particularly useful tool is the ability to "roll back" the windows to view the state of the program at any point. Finally, the structured knowledge base and graphics facilities allow the user to directly edit the knowledge base by simply adding and deleting links in graphs. In short, rather than directly specifying the details of *how* to solve the problem (by writing rules), the student or expert can show NEOMYCIN *what* the solution should look like.

4.3. Model of learning

GUIDON-DEBUG implements and extends the instructional paradigm proposed by Goldstein and Papert (Goldstein and Papert, 1977, Papert, 1980), particularly the concepts of "a glass-box expert," "an articulate learner," and "learning by debugging. The PSM and the procedural language of NEOMYCIN provide a special opportunity for formalizing a model of learning based on debugging, which enables us to relate learning, student modeling, and knowledge acquistion.

The student's process of debugging NEOMYCIN, generally called *knowledge acquisition,* is

similar to how a teacher might reason about a student. The approach is to account for the student's problem-solving behavior by inferring his knowledge and reasoning procedure. In debugging NEOMYCIN, the student asks the same questions GUIDON-DEBUG, the teacher, asks: "Why wouldn't I solve the problem that way? What do I know that he does not know?" The first-order approach is to transform this question to be, "What would I have to *change* in my model to solve the problem his way?" Thus, ODYSSEUS infers new domain relations that would be consistent with the student's annotated typescript. This approach of debugging by explaining has been demonstrated in knowledge acquisition (TEIRESIAS (Davis, 1976)), student modeling (MACYSMA ADVISOR (Genesereth, 1982a)), and learning (LEAP (Mitchell, et al., 1985)). Broadly speaking, the concept has also been called "failure-driven learning" (Schank, 1981).

By basing debugging on the *form* of a solution, we are describing a special kind of learning procedure. The problem solver can realize gaps in his knowledge, that is to focus on what we are attempting to teach him, by critiquing the form of the partial solution:

1. Solutions are models of a certain form (i.e., a graph, perhaps on multiple levels of detail, describing the process that caused the abnormal findings).

2. Problem solving moves forward, in part, by critiquing the form of the partial solution (i.e., gaps in the PSM).

3. Deficiencies of form are translated into incomplete reasoning processes (i.e., NEOMYCIN's model-manipulation tasks).

4. Failed or incomplete tasks are related to domain relations, which if present would have allowed the model transformation to take place (i.e., failed premises of metarules).

For example, given the observed deficiency of an unexplained finding under a secondary hypothesis, possible processes to resolve the deficiency are:

- Seek additional support for this hypothesis to make it the primary explanation.
- Attempt to rule out this hypothesis.
- Construct some explanation for the finding under the primary hypothesis.

Thus, the *form of the partial solution* suggests what to do: knowledge to apply, to infer, or to seek from an outside source. In this way, we can give the student a knowledge base that is *missing the facts we want to be sure he knows*, providing a context and tools that will enable

him to apply his background knowledge to detect and fill in the gaps. Even if he doesn't recall the missing facts, the *incorrect form* of the typescript or PSM will help the student ask good questions. For example, seeing that seizures is not connected to the most likely hypothesis, the student might ask, "Could meningitis cause seizures?" Or, "Could meningitis cause increased intracranial pressure?" In short, the program helps a student *realize what he does not know*, in a context that strongly motivates the need to learn. Our hypothesis is that using GUIDON-DEBUG will make the student aware that such a general critiquing procedure exists, and encourage him to use it when trying to understand his teacher or colleagues, or when reflecting on the state of his own problem solving (Schoenfeld, 1981, Barr, 1979b).

To summarize, the PSM can be used by the student in several ways:

- as an *explanation aid*, to determine NEOMYCIN's support for a particular hypothesis;
- as a *debugging aid*, to detect gaps in its diagnostic explanation; and
- as a *learning aid*, to detect new and relevant domain relations (and to a lesser degree diagnostic strategy) that he might want to study.

Furthermore, the PSM provides a basis for GUIDON-DEBUG to debug the student's knowledge, by detecting gaps that he introduces or fails to fill in. The PSM perspective also suggests a way of designing a curriculum for GUIDON-DEBUG. Bugs can be ordered according to difficulty, based on whether the findings and hypotheses mentioned by the missing rule appear in the PSM.

Finally, note that some of DEBUGGY's weak problem-solving strategies for resolving impasses in subtraction refer to the form of the subtraction notation (VanLehn, 1982). (For example, there should be no spaces between numbers in the answer.) Thus, our model of debugging is in general terms compatible with the idea of impasses and repairs. However, we explain repairs in terms of reasoning about the *correct procedure* and postulating new domain relations. In subtraction, the emphasis is reversed; the impasse generally occurs because of a gap in the procedure rather than the domain relations, the math facts.

5. Limitations

GUIDON-DEBUG is based on the idea that missing knowledge can be added by a student, avoiding the problem of a fixed tutor with one model of the world to teach. However, the forms of assistance the program can provide and what can be expressed in NEOMYCIN naturally limit its usefulness. These limitations fall into two broad categories:

- **Procedural abstraction** Extensions to ODYSSEUS to enable recognition of alternate strategies in terms of metarule modifications would require making explicit the pre-conditions and post-conditions of tasks, as well as the basis of metarule ordering (Clancey, 1984, Clancey, 1985). To make the meaning of tasks more explicit, we could also relate them to more abstract control constructs, such as "generators" and "filters."

- **Model semantics** To formalize the learning procedure (Section 4.3), to provide assistance to the student, additional causal knowledge must be represented for generating or justifying plausible PSM links. However, the amount of knowledge to be added is well beyond what we can consider at this time. This should be contrasted with formal or artifactual domains, such as programming and electronics, or those with much more circumscribed models such as geology, which are limited enough to be represented and used for automated learning (Genesereth, 1982b, Johnson and Soloway, 1984, Mitchell, et al., 1985, Smith, 1985).

Even within the range of what NEOMYCIN permits, much more can be done to use guidon-debug for studying knowledge acquisition and learning. We have run two types of experiments to date: (1) several medical students used GUIDON-WATCH, and (2) we collected protocols of students diagnosing someone acting as a patient. Further empirical studies are planned to investigate the advantages and limitations of GUIDON-DEBUG.

6. Conclusions

We have described an instructional program in which a student plays the role of knowledge engineer in improving a domain model, and presumably his own understanding, by debugging a knowledge-based program.

To recapitulate the main points:

- GUIDON-DEBUG exploits and develops AI methods for formalizing a problem-solving model, enabling reasoning to be studied and incrementally modified.

- Debugging focuses learning. It provides practical experience on actual cases, while reducing a large curriculum to manageable, incremental steps. Such a "workbench" approach gives more control to the student, avoiding the limitations and difficulties of developing an "omniscient tutor."

- The well-structured form of NEOMYCIN allows significant flexibility for reifying the reasoning process, contextually relating data requests to domain knowledge, and simplifying knowledge base editing.

- The idea of a patient-specific model makes concrete the explanation nature of diagnosis and the model-manipulation character of diagnostic tasks.

Much work remains to be done on evaluating the instructional benefit of the debugging scenario, providing tutoring assistance, enhancing the explanation capability, and extending the program's causal knowledge.

References

Anderson, J.R., Boyle, C.F., Yost, G. *The geometry tutor*, in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1-7, August, 1985. Volume 1.

Barr, A., Bennett, J., and Clancey, W. *Transfer of expertise: a theme for AI research.* Working Paper HPP-79-11, Stanford University, 1979.

Barr, Avron. *Meta-knowledge and Cognition*, in *IJCAI-79*, pages 31-33, February, 1979.

Brown, J. S. and Rubenstein, R. and Burton, R. *Reactive learning environment for computer-aided electronics instruction.* Technical Report 3314, Bolt Beranek and Newman, 1976.

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

Clancey, W.J., *An Antibiotic Therapy Selector Which Provides For Explanations.* Technical Report, Stanford Heuristic Programming Project, 1978. Working Note HPP-78-26.

Clancey, W. J. Applications-oriented AI research: Education. In Barr and Feigenbaum (editors), *The Handbook of Artificial Intelligence*, pages 223-294. William Kaufmann, Inc., Los Altos, 1982.

Clancey, W. J. GUIDON. In Barr and Feigenbaum (editors), *The Handbook of Artificial Intelligence*, chapter Applications-oriented AI research: Educationpages 267-278. 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. *The advantages of abstract control knowledge in expert system design*, in *Proceedings of the National Conference on AI*, pages 74-78, Washington, D.C., August, 1983.

Clancey, W. J. *Acq@uiring, representing, and evaluating a competence model of diagnosis.* HPP Memo 84-2, Stanford University, February 1984. (To appear in M. Chi, R. Glaser, and M. Farr (Eds.), *Contributions to the Nature of Expertise*, in preparation.).

Clancey, W. J. Representing control knowledge as abstract tasks and metarules. (To appear in *Computer Expert Systems*, eds. M. J. Coombs and L. Bolc, Springer-Verlag, in preparation).

Clancey, W.J. Qualitative student models. In L??? (editors), *Annual Review of Computer Science*, . Annual Reviews, Inc., 1986.

Clancey, W.J. The engineering of qualitative models. In preparation, to be submitted to AAAI-86.

Clancey, W. J. and Letsinger, R. NEOMYCIN: Reconfiguring a rule-based expert system for application to teaching. In Clancey, W. J. and Shortliffe, E. H. (editors), *Readings in Medical Artificial Intelligence: The First Decade*, pages 361-381. Addison-Wesley,

Reading, 1984.

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.

Dewey, J. The process and product of reflective activity: Psychological process and logical form. In R.D. Archambault (editor), *John Dewey on Education: Selected Writings*, pages 243-259. Random House, Inc., New York, 1964.

diSessa, A.A. The third revolution in computers and education. A report for the Committee on Mathematics, Science and Technology Education; Commission on Behavioral and Social Sciences and Education; National Academy of Sciences.

Genesereth, M. R. The role of plans in intelligent teaching systems. In D. Sleeman and J. S. Brown (editors), *Intelligent Tutoring Systems*, pages 137-155. Academic Press, New York, 1982.

Genesereth, M. R. *Diagnosis using hierarchical design models*, in *Proceedings of the National Conference on AI*, pages 278-283, Pittsburgh, PA, August, 1982.

Gentner, D. and Stevens, A. (editors). *Mental models*. Hillsdale, NJ: Erlbaum 1983.

Goldberg, A. *Computer-Assisted Instruction: The Application of Theorem-Proving to Adaptive Response Analysis.* Technical Report 203, IMSSS, Stanford University, 1973.

Goldberg. Educational uses of a dynabook. *Computers & Education*, 1979, *3*, 247-266.

Goldstein, I.P., and S. Papert. Artificial Intelligence, Language and the Study of Knowledge. *Cognitive Science*, 1977, *1(1)*, ???

Hasling, D. W., Clancey, W. J., Rennels, G. R. Strategic explanations in consultation. *The International Journal of Man-Machine Studies*, 1984, *20(1)*, 3-19. Also in Development in Expert Systems, ed. M.J. Coombs, Academic Press, London.

Hollan, J. D., Hutchins, E. L., and Weitzman, L. STEAMER: An interactive inspectable simulation-based training system. *The AI Magazine*, 1984, *5(2)*, 15-27.

Johnson, W. Lewis, and Elliot Soloway. *Intention-Based Diagnosis of Programming Errors*, in *Proceedings of the National Conference on AI*, pages 162-168, Austin, TX, August, 1984.

Langlotz, C.P. and Shortliffe, E.H. Adapting a consultation system to critique user plans. *International Journal of Man-Machine Studies*, 1983, *19(5)*, 479-496.

Miller, P.L., M.D., Ph.D, Anger, D., M.D., Marks, M.S., Sudan, M.D., and Tanner, G., M.D. *Teaching with "attending": A practical way to "debug" an expert knowledge base*, in *Proceedings of the AAMSI Congress 83*, pages 87-91, Bethesda, 1983. Volume 1.

Mitchell, T.M., Mahadevan, S., Steinberg, L.I. *LEAP: A learning apprentice for VLSI design*, in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 573-580, Los Angeles, August, 1985.

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

Patil, R. S., Szolovits, P., and Schwartz, W. B. *Causal understanding of patient illness in medical diagnosis*, in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 893-899, August, 1981.

Reiser, B.J., Anderson, J.R., and Farrell, R.G. *Dynamic student modelling in an intelligent tutor for lisp programming,* in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence,* pages 8-14, August, 1985. Volume 1.

Richer, M.H. andd Clancey, W.J. GUIDON-WATCH: A graphic interface for viewing a knowledge-based system. *IEEE Computer Graphics and Applications,* November 1985, *5(11),* 51-64.

Rouse, W.B. and Morris, N.M. *On looking into the black box: prospects and limits in the search for mental models.* Technical report 85-2, School of Industrial and Systems Engineering, Georgia Institute of Technology, May 1985.

Schank, R. C. Failure-driven memory. *Cognition and Brain Theory,* 1981, *4(1),* 41-60.

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.

Sleeman, D. and Brown, J. S. (editors). *Intelligent Tutoring Systems.* New York: Academic Press 1982.

Smith, B. *Models in expert systems,* in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence,* pages 1308-1309, 1985.

Stevens, A. L. and Collins, A. *The goal structure of a Socratic tutor.* Technical Report 3518, BBN, 1977.

Stevens, A. L., and Collins, A. Multiple conceptual models of a complex system. In Snow, R., Federico, P., and Mantague, W. (editors), *Aptitude, Learning and Instruction: Cognitive Process Analysis,* . ???, 1978.

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.

Thompson, T. and Clancey, W. J. A qualitative modeling shell for process diagnosis. *IEEE Software,* March 1986, *3(2),* 6-15.

VanLehn, K. *Empirical studies of procedural flaws, impasses, and repairs in procedural skills.* Technical Report ONR-8, LRDC, University of Pittsburgh, March 1982.

Wenger, Etienne. *Knowledge communication systems, an artificial intelligence approach to computer-aided instruction.* Los Altos, CA: Morgan and Kaufman Inc. 1986.

Wilkins, D.C., Clancey, W.J., and Buchanan, B.G. An overview of the Odysseus learning apprentice. In T.M. Mitchell, J.G. Carbonell, and R.S. Michalski (editors), *Machine Learning: a Guide to Current Research,* . Academic Press, 1986.