

Context Dependent Planning  
In A Machine Tutor

Beverly Park Woolf

COINS Technical Report 84-21

Department of Computer and Information Science  
University of Massachusetts  
Amherst, Massachusetts, 01003  
May 1984

**CONTEXT DEPENDENT PLANNING  
IN A MACHINE TUTOR**

**A Dissertation Presented**

**By**

**BEVERLY PARK WOOLF**

**Submitted to the Graduate School of the  
University of Massachusetts in partial fulfillment  
of the requirements for the degree of**

**DOCTOR OF PHILOSOPHY**

**May 1984**

**Computer and Information Science**

© Beverly Park Woolf 1984  
All Rights Reserved

**This work was supported in part by**

**The Department of Computer and Information Sciences  
University of Massachusetts, Amherst, Mass.**

**The United States Army Research Institute  
for the Behavioral and Social Sciences  
Grant Numbers, MDA903-80-C-0508 and DAHC 19-77-G-0012**

**In Loving Memory of**

**Pearl Katz Park**

**Irving Herman Park**



## ACKNOWLEDGEMENTS

My intellectual debt to my committee is enormous. David McDonald, especially, has been a tireless director, collaborator, and friend. He provided clear and concise suggestions, unmitigated goading, and gentle prodding that moved me always toward the next unsolved problem. He gave me a sense of the excitement and responsibilities inherent in a research project. David has an uncanny sense for what is involved in good writing and helped both with the detailed line editing and the larger structural problems. He put extraordinary hours and care into both the development of the research and the production of the document.

Michael Arbib, a brilliant thinker and a rich personal source of inspiration, relentlessly encouraged me. He posed some of the more difficult questions, had the insight not to answer them, and always added some perspective on how to approach their solution. Michael made many careful suggestions to improve both the structure and readability of this document.

Edwina Rissland provided elegant competence and unflagging emotional and intellectual support. By her example, I learned the importance of diligence and preparation in pursuit of research goals. She and Michael taught some of the best courses I took in my graduate career.

Elliot Soloway was my original mentor, inspiration, and once lonely believer in me. He relentlessly cheered me on from the start and convinced other to do the same.

Howard Peelle supported my early efforts and shared the motives behind this work. He had the wisdom, at the beginning, to allow me to structure and pursue the research in my own way.

I give special recognition to Edward Riseman, chairperson of the department, for his personal support of me and my research during the last five years. By his energy and intellect, he has pioneered a rapidly growing and cohesive group of students, faculty, and associates who work together to create remarkable results. Ed has established an unparalleled environment for intellectual growth and management of research projects. He is motivated by an uncanny, some would say "nonrational," vision of how an effective organization is run and how research is conducted. His record as chairperson suggests that mortals can move mountains.

Thanks to Stephen Woolf who designed and produced the graphics in the dissertation. He put many hours into understanding the system, representing its workings, and drafting the final figures.

I thank especially several individuals whose friendship and emotional guidance were critical to my development. Steve Levitan has been my intellectual companion, toughest critic, and computer pal for 5 years. He worked long and hard hours with me as we both put together dissertations, frequently recognizing the rough parts of the work before

I did and rallying me back to some form of equilibrium.

My companions and close friends, Jeff Bonar and Jeff Conklin, helped me learn several intellectual and spiritual lessons. Don House supported me and allowed me to test out my meager ideas with him where others would have blanched. Victor Lessor provided stimulating guidance and direction during my early work and Dan Corkill, by his equanimity and programming skills, provided hope and reassurance, in addition a superb home-grown LISP system.

Other supported me in diverse ways: Susan Parker performed careful editing and typing, often providing overnight service when necessary; Jody Simpson provided indepth editing and made substantive improvements to the document; Rinate Ridders, Elsjie Sturtevant, and Dade Singapuri inspired me to exercise and dance; and Werner Erhart and Fernando Flores helped me maintain a clear sense of self and purpose through their training programs.

I acknowledge the loving support of Abraham and Ruth Woolf, who offered the challenge of their general disbelief and ultimately their support and guidance for the unorthodox processes behind this work.

Thanks to Ellie and Luna Woolf for learning, growing and taking care of business while I wrote this dissertation. Thanks to Stephen for remaining loyal, steadfast and for nuturing me. They each kept a family together against great odds.



ABSTRACT

CONTEXT DEPENDENT PLANNING

IN A MACHINE TUTOR

May, 1984

Beverly Park Woolf, B.A., Smith College

M.S., Ph.D., University of Massachusetts

Directed by: Professor David D. McDonald

This thesis describes Meno-tutor, a computer program for tutoring. The program adapts its utterances to the context of student and discourse history, i.e., it responds differently to the knowledgeable student and the confused one. The program uses knowledge about tutoring strategies, complex communication skills, and its ability to infer the level of the student's knowledge to generate reasonable tutoring discourse. It can question a student about the subject domain, probe him for possible misconceptions, and can change its strategies if the student is not progressing well.

The planning mechanism for Meno-tutor is best described as a set of decision-making states organized into three levels. Each state provides a constraint or decision about the form and content of the utterance. The levels successively define the actions of the next level and constrain the pedagogical, strategic and tactical formation of the utterance.

States are linked to other states by a structure which is nominally an OR graph, but makes two notable deviations. The first is a set of default transitions that represent crosslinks through the states and are responsible for the traditional sections of discourse, e.g., introduction of a topic, questioning the student about the topic, and terminating that topic. The second is a set of "meta-rules" which functionally represent the shifts observed in classic human tutoring; they express the high-level transitions characteristic of, for example, a change in strategy. The path of the tutor can be preempted before entrance to any state by meta-rules.

Meno-tutor is an attempt at a "generic" tutor, i.e., one not committed by design to a single subject. It teaches about the causal reasoning behind rainfall and the looping constructs in the programming language Pascal. In both subject areas it adjusts its response to its own inferences about the student's level of knowledge and to the success of the discourse.

## **A Guide to the Reader**

The computer scientist reading this document might be most interested in the discourse analysis and system design in Chapter 3, the implementation in Chapter 4, and the results in Chapter 5. The researcher in cognition and teaching might be most interested in the philosophy of tutoring in Chapters 1 and 6, and the design of a tutoring system in Chapter 2. Section 6.6 is a special section for educators, describing the implications of this and similar research for education and providing pointers to the relevant parts of the dissertation.

## TABLE OF CONTENTS

|                              |          |
|------------------------------|----------|
| <b>LIST OF FIGURES</b> ..... | <b>x</b> |
|------------------------------|----------|

---

### Chapter

|   |           |
|---|-----------|
| <b>I. INTRODUCTION</b> .....  | <b>1</b>  |
| <b>1.1 Motivations for this Work</b> .....                              | <b>1</b>  |
| <b>1.2 The Major Contributions</b> .....                                | <b>3</b>  |
| <b>1.3 Development of a Philosophy of Tutoring</b> .....                | <b>4</b>  |
| <b>1.4 Design and Implementation of a Generic Tutor</b> .....           | <b>6</b>  |
| 1.4.1 Observations of Human Tutoring .....                              | 6         |
| 1.4.2 Design Issues in the Implementation of a Machine Tutor .....      | 8         |
| 1.4.3 Information about the System .....                                | 8         |
| 1.4.4 Information about the Student .....                               | 9         |
| <b>1.5 An Initial Exploration of the Tutor</b> .....                    | <b>9</b>  |
| <b>II. DESIGN ISSUES IN BUILDING A TEACHING SYSTEM</b> .....            | <b>13</b> |
| <b>2.1 Pre-A.I. Instructional Systems</b> .....                         | <b>13</b> |
| <b>2.2 Intelligent Computer Aided Instructional Systems</b> .....       | <b>14</b> |
| <b>2.3 The Components of a Teaching System</b> .....                    | <b>16</b> |
| <b>2.4 Expert Module</b> .....  | <b>19</b> |
| 2.4.1 Grain Size of Knowledge .....                                     | 19        |
| 2.4.2 Examples of the Grain Sizes of Teaching Systems .....             | 21        |
| 2.4.3 The Language of the Expert Module .....                           | 24        |
| 2.4.4 Examples of Languages in the Expert Module .....                  | 24        |
| <b>2.5 Student Model</b> .....  | <b>25</b> |
| 2.5.1 Overlay and Skill Modelers .....                                  | 26        |
| 2.5.2 Examples of Overlay and Skill Modeling .....                      | 26        |
| 2.5.3 Bug Modelers .....  | 28        |
| 2.5.4 Examples of Bug Modeling .....                                    | 28        |
| <b>2.6 Teaching Component</b> .....                                     | <b>29</b> |
| 2.6.1 Behaviorist Learning .....  | 30        |
| 2.6.2 Constructivist Learning .....                                     | 30        |
| 2.6.3 Examples of Constructivist Systems .....                          | 31        |
| 2.6.4 Wet Sponge Learning .....   | 32        |
| <b>2.7 Communication Module</b> .....                                   | <b>32</b> |
| 2.7.1 Examples of Communication Modules .....                           | 33        |
| <b>2.8 Summary</b> .....  | <b>34</b> |
| <b>III. DISCOURSE ANALYSIS AND THE DESIGN OF A COMPUTER TUTOR</b> ..... | <b>35</b> |

|  |   |            |
|--|---|------------|
| 3.1  | Understanding Discourse . . . . .   | 35         |
| 3.1.1  | Complex Speech Acts . . . . .   | 38         |
| 3.1.2  | Discourse Management . . . . .  | 40         |
| 3.1.3  | Alternative Dialogues . . . . .   | 43         |
| 3.2  | Design of the Computer Tutor . . . . .                                      | 45         |
| 3.2.1  | Levels of Planning . . . . .  | 46         |
| 3.2.2  | Representing Discourse by States . . . . .                                  | 48         |
| 3.2.3  | The Annotated Knowledge Base . . . . .                                      | 51         |
| 3.3  | Summary . . . . .   | 54         |
| <b>IV. RULES TO DETERMINE DISCOURSE TRANSITION . . . . .</b> |   | <b>55</b>  |
| 4.1  | The Design of the Tutoring Processes . . . . .                              | 55         |
| 4.2  | The Control Structure of Meno-tutor . . . . .                               | 56         |
| 4.2.1  | Context-Independent Transitions . . . . .                                   | 56         |
| 4.2.2  | Context-Dependent Transitions . . . . .                                     | 57         |
| 4.3  | An Example Discourse . . . . .  | 70         |
| 4.3.1  | Explicit Incorrect Acknowledge Tactic . . . . .                             | 70         |
| 4.3.2  | Teach Data Strategy . . . . .   | 74         |
| 4.3.3  | The Explore Strategy . . . . .  | 76         |
| 4.3.4  | The Implicit Acknowledge Tactic . . . . .                                   | 79         |
| 4.4  | Discussion of the rules . . . . .   | 82         |
| 4.4.1  | Mechanisms for Authoring rules . . . . .                                    | 82         |
| 4.4.2  | Editing a rule . . . . .  | 83         |
| 4.5  | Summary . . . . .   | 83         |
| <b>V. RESULTS AND EXAMPLES . . . . .</b>                     |   | <b>85</b>  |
| 5.1  | The Variety of Utterances . . . . .   | 85         |
| 5.2  | Discourses Produced by Extending Meno-tutor . . . . .                       | 89         |
| 5.2.1  | Varying Meta-Rules . . . . .  | 89         |
| 5.2.2  | Varying the Expert Knowledge Base . . . . .                                 | 91         |
| 5.2.3  | Varying Michelin Ratings . . . . .  | 94         |
| 5.2.4  | Working with the Knowledgeable Student . . . . .                            | 97         |
| 5.3  | Summary . . . . .   | 98         |
| <b>VI. CONCLUSIONS . . . . .</b>                             |   | <b>101</b> |
| 6.1  | A Review of the Goals . . . . .   | 101        |
| 6.2  | An A.I. Theory of Tutoring . . . . .  | 102        |
| 6.2.1  | Tutoring as a Function of Language Generation . . . . .                     | 102        |
| 6.2.2  | Requirements for Style . . . . .  | 103        |
| 6.2.3  | Requirement for and Sufficiency of Preemption Based on<br>Context . . . . . | 104        |
| 6.2.4  | Requirements for Feedback . . . . .   | 106        |
| 6.2.5  | Absence of a Requirement for a Recursive Stack . . . . .                    | 106        |
| 6.3  | Features of the System . . . . .  | 107        |

|  |   |            |
|--|---|------------|
| 6.4  | Limitations of the System . . . . .                 | 110        |
| 6.5  | Implications of the Research . . . . .              | 111        |
| 6.6  | Implications of the Research to Education . . . . . | 111        |
| 6.7  | Further Research Directions . . . . .               | 115        |
| <b>VII. BIBLIOGRAPHY . . . . .</b>                                   |   | <b>117</b> |
| <hr/>  |   |            |
| <b>APPENDIX A. TRACES OF THE OPERATIONS OF THE MENO-TUTOR . . .</b>  |   | <b>125</b> |
| <b>APPENDIX B. LISP PRETTY PRINTING OF THE TUTORING STATES . . .</b> |   | <b>133</b> |
| <b>APPENDIX C. HIGH LEVEL CODE OF THE MENO-TUTOR . . . . .</b>       |   | <b>141</b> |
| <hr/>  |   |            |

## LIST OF FIGURES

|  |    |
|--|----|
| 1. Weather Dialogue [Stevens et al., 1982] . . . . .                     | 7  |
| 2. The Second Meno-tutor Discourse . . . . .                             | 10 |
| 3. The Pascal Discourse . . . . .  | 11 |
| 4. Selected Teaching Systems . . . . .                                   | 15 |
| 5. The Environment of a Teaching Machine . . . . .                       | 17 |
| 6. Criteria for Building Intelligence into a Teaching System . . . . .   | 18 |
| 7. An Example of SPADE-0 Interaction . . . . .                           | 22 |
| 8. Differences in the Speaker's Goal . . . . .                           | 37 |
| 9. Common Speech Patterns and Strategies in Discourse . . . . .          | 39 |
| 10. Discourse to Handle Unspoken Problems . . . . .                      | 41 |
| 11. Discourse Management Diagram (DMN) . . . . .                         | 44 |
| 12. The Rain Dialogue . . . . .  | 45 |
| 13. The Tutoring States . . . . .  | 49 |
| 14. Three Typical States at the Tactical Level . . . . .                 | 50 |
| 15. The Expert Knowledge Base for the Rainfall Discourse . . . . .       | 52 |
| 16. The Action of Meta-Rules . . . . .                                   | 58 |
| 17. Informal Notation of a Meta-Rule . . . . .                           | 59 |
| 18. Application of a rule . . . . .                                      | 60 |
| 19. Pretty Printing of the Code for Pedagogic Meta-rules . . . . .       | 61 |
| 20. A Description of the Pedagogic Meta-Rules . . . . .                  | 62 |
| 21. Pretty Printing of the Code of the Strategic Meta-Rules . . . . .    | 63 |
| 22. A Description of the Strategic Meta-Rules . . . . .                  | 64 |
| 23. Pretty Printing of the Code for the Tactical Meta-Rules . . . . .    | 65 |
| 24. A Description of the Tactical Meta-Rules . . . . .                   | 67 |
| 25. The Discourse Snippet . . . . .                                      | 70 |
| 26. The Portion of the Tutoring Structure used in the Example . . . . .  | 71 |
| 27. Explicit Acknowledgement Snapshot . . . . .                          | 72 |
| 28. Code for Meno-tutor and The Controller . . . . .                     | 72 |
| 29. Code for Process Node . . . . .                                      | 73 |
| 30. The Code for Pop Planning Level and Evaluate State . . . . .         | 74 |
| 31. The High-level Algorithm for Generating Tutoring Discourse . . . . . | 75 |
| 32. Teach to Teach Specific Snapshot . . . . .                           | 76 |
| 33. Teach to Explore Snapshot . . . . .                                  | 77 |
| 34. Explore to Exploratory Question Snapshot . . . . .                   | 78 |
| 35. Explicit to Implicit Incorrect Acknowledgement . . . . .             | 79 |
| 36. Implicit to Describe Snapshot . . . . .                              | 80 |
| 37. Describe Strategy Snapshot . . . . .                                 | 81 |
| 38. Utterances to Teach about Climate . . . . .                          | 87 |
| 39. Utterances to Teach about Rainfall . . . . .                         | 88 |
| 40. The Second Rainfall Discourse . . . . .                              | 90 |
| 41. A Student Pascal Program . . . . .                                   | 92 |
| 42. Meno-tutor Discussing the Student's Errors . . . . .                 | 93 |
| 43. The Expert Knowledge Base for the Pascal Discourse . . . . .         | 94 |
| 44. The Rainfall Knowledge Base with new Michelin Rating . . . . .       | 95 |



|   |     |
|---|-----|
| 45. Discourse Produced by Varying Michelin Rating . . . . . | 96  |
| 46. Discourse with the Knowledgeable Student . . . . .      | 98  |
| 47. A Bad Discourse . . . . .                               | 104 |
| 48. Goals Used by Human Tutors . . . . .                    | 104 |

# CHAPTER I

## INTRODUCTION

This research focuses on the knowledge that enables a tutor to engage in a discourse with a student. It defines the components of such knowledge and proposes computational processes that are used to implement a prototype "generic" tutor. Meno-tutor is the computer tutor built as a part of this research. It adapts its discourse to the context of student and discourse history, i.e., it responds differently to the knowledgeable student and the confused one. It teaches in two different domains and predicates its choice of topics on the an importance annotation in the knowledge base. The tutor uses knowledge about tutoring strategies, complex communication skills, and its ability to infer the level of the student's knowledge to generate reasonable tutoring discourse.

The discourse produced by the prototype tutor is said to be "context-dependent," i.e., the output behavior of the machine is different in different contexts. Text is generated only after consideration of previous interaction and in relation to the nature of student knowledge and expert knowledge. For instance, unimportant subject matter, or data which is trivial in the context of a student's observable knowledge, is not given the same emphasis as critical data.

The research confronts such issues as what is involved in maintaining flexible discourse, how a discourse can remain coherent, and what output makes reasonable sense in light of the student's prior actions. The research defines some elements in a theory for tutoring (see Chapter 6) and addresses some theoretical issues in the implementation of a system for machine discourse.

### 1.1 Motivations for this Work

The research is motivated by three long-term goals: (1) to make computers more conversant, (2) to *build expert teaching machines*, and (3) to *understand discourse*. The first motivation, to make computers more conversant, is prompted by the advent of sophisticated systems, coupled with the growth of powerful and inexpensive computers. These technological advances suggest that computers will increasingly be made available to, and will need to be extended by, the average user. This user will probably be inexperienced in programming and will almost certainly lack specific training for each particular system. Each system, therefore, will have to explain its own problem solving behavior and will have to be able to observe errors in the user's interaction with the system.

By and large, today's systems cannot do this (see below for the notable exceptions). Because these systems lack an understanding of their own reasoning in addition to the requisite communication skills, many large systems are out of the reach of the average person. Some of these systems are even out of reach of the expert because the particular syntax or semantics is not fully explained.

When computer systems become commonly available and easily extensible, they will need to communicate and to explain their own reasoning. They will also need to understand the user's knowledge, clarify his goals, recognize his difficulties, and communicate at a level that he can understand. In other words, these systems will need to have the skills and abilities of an expert tutor. They will need to respond differently to different users and to discuss topics that are relevant to the individual user. To develop such communication skills in a system requires research into the processes of discourse and tutoring. It requires an understanding of what is special about tutoring and of what enables a tutor to make inferences about the student's knowledge. This research provides an initial exploration of these issues and offers some solutions.

Research has been done on generating limited explanations or isolated responses from an expert systems or large data base [McKeown, 1980; Clancey, 1979a; Swartout, 1981]. Explanations have been produced graphically [Forbus & Stevens, 1981]. However communication in these systems is typically based on the stereotypic model of the user; responses are not adjusted to the particular user. The new user requires explanations that are distinct from those for the expert; the novice might need an explanation about basic system interactions, or how the data is stored; whereas the expert might be satisfied with a listing of permitted command syntax. The average user will need a response somewhere between the two extremes -- a response tailored to his own level of competence in working with the system.

A second motivation for this research is the desire to build expert teaching machines. Adapting an expert system to a teaching machine is not yet a straightforward process [Clancey, 1979a]; it is, however, a reasonable goal. If we can make the knowledge of an expert system available to the student and learn how to build sophisticated software for tutoring, we can potentially create a variety of teaching machines of a variety of subjects. The potential is to make excellent teaching machines commonly available.

Current expert systems are, for the most part, unable to explain their own knowledge. Large, complex expert systems (MYCIN [Shortliffe, 1976], INTERNIST, [Pople, 1977]) routinely make decisions based on complex decision rules. Yet the information associated with these rules is not sufficient to explain the reasoning of the system. The rules are abbreviated because the rationale behind them is not necessary for the machine's reasoning. Often the reasoning is not consciously available to the expert at the time the system is built. Even though these rules provide enough information for the expert or rule-writer to build fundamental problem solving into the system for that domain, they are not sufficient to explain the problem solving to the novice or to the next author of the system. This deficiency is due to the fact that the

rules were stripped of organizational, causal, or structural data. Such abbreviations of data are used to construct the system. Yet, in order for an expert system to explain itself, the rules must be recast with the structural and causal knowledge which was once removed.

A third, yet possibly the greatest, motivation for this research is the desire to understand discourse and tutoring. Though teaching, explanation, and verbal guidance occur in many domains and under many rubrics, we do not seem to understand what makes one discourse effective and memorable and another weak and ineffectual. It is possible that the development of machine tutoring may provide validation for some tutoring theories in education and psychology just as cognitive science may provide a basis for work in linguistics and information processing psychology.

## **1.2 The Major Contributions**

The purpose of this research is to explore the issues around discourse and teaching. In the previous section, I set forth the long term goals of this body of research. In this section, I list the progress I have made toward solution of the more short term goals. One contribution of this research is the creation of a precise theory of tutoring. This theory will be motivated in the next section and elaborated in Chapter 6. Other contributions lie in the areas of system design and of the cognitive processes underlying tutoring. I shall show that instead of being a simple iterative exploration of topics from a stored representation of data, tutoring requires consideration of levels of planning knowledge that organize and focus the presentation of topics to a student. These contributions and the results reported in Chapter 6 have immediate application in the design of tutoring systems and long-range application to the next generation of discourse and interface systems.

The major contributions can be grouped into three categories:

1. Development of a philosophy of tutoring.
2. Design and implementation of a generic tutor.
3. An initial exploration of the tutor.

These contributions will be described in the following three sections.

### 1.3 Development of a Philosophy of Tutoring

Tutoring, as a form of communication, suffers from the problems that afflict other forms of communication: the tutor cannot know with certainty whether the student understands what he is saying. Further, the tutor can not know what meaning to attribute to the student's answers. The student's knowledge will never be completely understood by the tutor and so he must continually suggest new topics or alternative viewpoints that might trigger the correct answer or probe the wrong one. In addition, the student is not aware of what he *does not know*, so the tutor, more than the typical speaker, must take care to make clear for both participants what information is missing and what information is erroneous. (Chapter 3 presents a detailed discussion of this phenomenon.)

Tutoring is a linguistic exchange whose goal is to clarify a body of knowledge to which the student has already been exposed (e.g., through lectures or reading). It involves directing a dialogue so that it is responsive and flexible even in the face of errors. The words spoken between student and teacher are only the tip of the iceberg of what they know about each other and what they are communicating in the dialogue; in fact, the dialogue must be predicated on a deep understanding of the student's viewpoint and must incorporate a variety of sophisticated communication skills that probe and correct the student's knowledge. In order to design a communication system that produces this kind of result, the essential activities that characterize the tutoring phenomenon must be analyzed. To this end, we must recognize those aspects of the tutoring discourse that are effectual and those which are not [Winograd & Flores, to appear].

As an example of a machine which captures some of the essential qualities of what was once a non-mechanical phenomenon, I present the automobile. The design of the automobile captured some of the essential aspects of the phenomenon of transportation, e.g., comfort, speed, and safety. It did not reproduce the actual activities of transportation -- the horse and cart for instance. Rather, it modelled the essential aspects of transportation. By not copying from the expression of the phenomenon of that day, the automobile was able to increase on the activities of transportation many-fold as compared to the days of the horse and cart.

A word processor, too, is an example of a machine which enhances another human activity -- writing. The design of the word processor captures the functionality of text production; it does not, for instance, interpret the phenomenon within the narrow confines of secretarial skills, and include activities such as blocking out titles, aligning margins, or setting tabs. Rather, the word processor introduces activities like random text access and dynamic "cut and paste" options. These activities capture the functionality of the phenomenon and -- like the automobile -- enable the automated system to increase the functionality of the original phenomenon manyfold.

In an effort, then, to design a machine tutor, I need to clarify the phenomenon of tutoring. To date, tutoring has been known only in the context of one-on-one dialogues between human teachers and students. These dialogues often take place in classrooms where 20-40 students are present. To create the machine tutor I need to recognize two factors: 1) *what tutoring is* apart from the way it is practiced in its present context, and 2) *what tutoring does*.

Tutoring is a linguistic exchange between teacher and student. It is a special form of teaching whose goal is to extend the student's powers of thinking and acting in the subject area. Personal observation of human tutoring, as well as analysis of the protocols described in Chapter 3, suggest that, in addition to the linguistic exchange, several implicit (nonlinguistic) aspects of the exchange are essential to the phenomenon. These aspects include *transparency, trust, guidance from a hidden agenda, and communication skills*. Each aspect will be discussed below.

An expert tutor acts *transparently*; he seems to be able to affect the student's knowledge and to clarify it without the student noticing how the probe or the correction takes place. This transparency may not be apparent in the context of classroom tutoring where activities such as pretests, posttests, drill and practice exercises, or graduated workbooks have taken on the definition of tutoring. These activities are interpretations of tutoring in the context of the classroom. They have developed there because the ratio of students to teacher is large and the kind of one-on-one sensitive and invisible tutor which I have been describing has not been possible. Classic tutoring activities as we know them from the classroom may not play any role in a machine tutor; they might simply be artifacts of the context of tutoring today.

The machine tutor which functions transparently will be more like a partner than an autocratic teacher [Pelle & Riseman, 1975]. The more successful the machine and the more effective it becomes, the more its explicit activities, such as questioning or correcting, should become transparent. This has happened with the word processor and automobile: the more sophisticated the machine, the more its functionality has increased while its explicit activities have disappeared.

Tutoring seems to be based on *trust*. The student trusts that the human tutor will provide a supportive environment within which the student can express his beliefs. By a supportive environment, I mean one with a positive cast, one which encourages innovation and experimentation. The focus of a supportive environment is to build and enrich the student's knowledge, rather than to correct his mistakes or constrain him into the "correct" behavior. In the supporting and trusting relationship, the tutor helps the student sort out his own failings, while using fairly-well defined techniques such as supplying missing errors in order to exhibit his fallacious reasoning.

The expert tutor seems to be guided by a *hidden agenda* of topics to discuss and misconceptions to explore. The tutor often follows this agenda until some statement or behavior by the student leads him to change his view of the student's knowledge. The tutor's ability to probe the student's knowledge and to unravel his misunderstandings

provides one of the greatest opportunities for the tutor to contribute to the student's knowledge because it eliminates conceptual problems that might interfere with his ability to learn.

The practical aspects of tutoring involve explicit *communication skills*, which provide some desiderata for design of a mechanical tutor. These skills include, but are not limited to the ability to:

- Understand the student and account for his knowledge or lack of it;
- Tailor the discourse to correspond to the student's knowledge, which is unraveled as the discourse progresses;
- Present appropriate tasks pitched at the student's level of understanding;
- Maintain coherency in the discourse and discuss topics that are understood by, and important to, the student.

The research described in this dissertation explores the use of the computer for each aspect of tutoring described above and proposes some solutions.

## 1.4 Design and Implementation of a Generic Tutor

There are several interesting problems in designing a generic tutor – one which can teach a student in a variety of ways about multiple subjects. These problems focus on two broad issues: (1) how can the system determine what the student knows, and (2) how can the system produce coherent discourse in the face of confused or limited student knowledge.

### 1.4.1 Observations of Human Tutoring.

In this research, I examined a dozen human tutoring dialogues in an attempt to learn techniques used by human tutors (see Chapter 3). As an example of a tutoring discourse I present a protocol from research by Stevens, Collins and Golden [1982, pg 19] in Figure 1. In this discourse, the student corrected his responses midway through the discourse: his first two answers were incorrect, yet his last two responses were correct. I do not suggest that the student "learned" during this interchange; that is a debatable and unresolvable issue. I do suggest, however, that the tutor's ability to manipulate his own tutoring style and to switch the topics of the domain was, in part, responsible the student going from the wrong answers to the right ones.

*Tutor:* Do you know what the climate is like in Washington and Oregon?

*Student:* Is it cold?

*Tutor:* No, it is rather mild. Can you guess about the rainfall there?

*Student:* Normal, I guess.

*Tutor:* Well, the Japan current, which starts in south eastern Pacific, goes along the coast of Japan and across the North Pacific ending up off the coast of Washington and Oregon. How do you think that current affects the climate there?

*Student:* It's probably rainy.

*Tutor:* It is indeed very wet, there are rain forests in both Washington and Oregon. What does that tell you about the temperature of the Japan current?

*Student:* It's warm.

**Figure 1: Weather Dialogue [Stevens et al., 1982].**

---

The modeling question, then, for the designer of a tutoring system is how to generate the strategies and structure observable in this example. I suggest that one of the techniques used by the tutor is careful management of the discourse within a space of several options. The tutor has the option, for instance, to ask questions, to describe new information, or to ask questions about topics and about dependencies between the topics. Above all, the tutor is free to abandon an approach which does not seem to work in favor of another one which may help the student to deduce the correct answer. In this example, the student was able to infer the facts about the rainfall in Washington and Oregon after learning about the Japan Current.

Another technique is to order topics: in the example above, the tutor chose from among a variety of topics: first he talked about climate and rainfall, then about the Japan Current, then back to climate again, and finally about the temperature of the Japan Current.



### 1.4.2 Design Issues in the Implementation of a Machine Tutor.

The knowledge used to build a machine tutor can be divided in two general categories: knowledge to handle the data in the system and knowledge to handle the student. The first deals with the storage, retrieval, and management of data from multiple knowledge bases. It includes information about the subject area, the rules of discourse, and common learning strategies. The second category includes information about how to evaluate and motivate the student. This information is not particular to student, but uses the student's answer to reason about him and infer an appropriate level at which to address the communication.

In the following sections I outline several of these issues and some of the results I have achieved.

### 1.4.3 Information about the System.

The first set of issues concerns representing, storing and discussing topics in the subject area. This includes knowledge of how to tutor and how to separate tutoring information from domain knowledge and how from the program itself. A decision must be made, for instance, about whether extra information about how to tutor is needed for a particular domain or whether canonical tutoring knowledge is sufficient and generalizable to the new domain.

In this research, I separate from the basic program, information about the domain and about tutoring. Information from each source of knowledge is maintained separately, yet elements from one source may be interleaved with information from another in a particular instance. The three knowledge sources are:

*Tutoring knowledge.* This kind of knowledge includes the strategies, rules, and processes that govern the system's interactions with the student. It includes how to tutor, what tools to use, and how often to use them.

*Domain and misconception knowledge.* This knowledge includes both correct and misconception information about the subject area. I have implemented two knowledge bases: 1) information about rainfall in Washington and Oregon and 2) elementary PASCAL data including links between incorrect answers and possible misconceptions.

*Text planning knowledge.* This source of knowledge holds the data and processes needed for communication. It includes knowledge about how to describe or explain a topic and how to question the student about a topic. It also includes knowledge about how to probe for mistakes. The system must arbitrate among several methods of communication and know, for instance, when and how to apply each method. I have implemented a set of planning templates that makes choices between alternative discourse interactions with the student.

#### 1.4.4 Information about the Student.

The second set of implementation issues concerns evaluating the student's knowledge and communicating with him at an appropriate level. To accomplish this the system must:

*Model student knowledge* using student input to formulate a representation of knowledge. I chose to represent the student's knowledge as annotations on a data base of correct and misconception knowledge.

*Remain flexible* and tailor its response to what the student actually says rather than sticking strictly to its original model of his knowledge. It should adjust what it says to be consistent with a dynamic model of the student. I use a preemptable control mechanism (see Chapter 4) to plan output at the level of the student's understanding.

*Maintain coherence in the discourse* and choose topics that can be understood by the student. To do this, the system must recognize the relationship between topics, especially as the student views these topics. I represent knowledge of the domain by a rich knowledge base built from empirical studies [Stevens et al., 1982; Soloway et al., 1981] of how students learn in the domain.

### 1.5 An Initial Exploration of the Tutor

Meno-tutor is the system implemented as a part of this research. Its name, in keeping with Soloway et al., [1981c], acknowledges my indebtedness to Plato [1922] who defined an instructional methodology where learning is as a process of clarifying and repairing errors. In Plato's essay, Meno, a slave boy learned the laws of trigonometry under the tutelage of Socrates. Socrates concluded that the slave boy actually knew the facts of trigonometry and that, he, the master teacher, simply facilitated the process by which this knowledge was revealed to the slave boy.

In a similar way, the goal of Meno-tutor is to reason about the student's knowledge and to adapt what it says to reveal that knowledge. The system probes the student about possible misconceptions or topics in the domain and changes its approach based on its deductions about the level of the student's knowledge. The tutor reasons about tutoring strategies, the quality of the discourse, and multiple knowledge bases before making decisions about the direction of the discourse.

At first, human tutoring discourses were used to "reverse engineer" Meno-tutor. Twelve human discourses, along with the one presented in Figure 1, were used to study the knowledge and structures required in tutoring discourse. Meno-tutor reproduced the discourse in Figure 1 generating the identical text and discourse transitions observed in the original human discourse.

Later, the system was extended to another domain and two other tutoring styles. For instance, the discourse in Figure 2 was produced by changing the preconditions of the rules that triggered the context-dependent transitions. The overall structure and emphasis of the second discourse are different, yet the the topics and knowledge base remain essentially the same.

---

- Tutor:* Do you think it rains much in Oregon?
- Student:* No.
- Tutor:* Why do you think it doesn't rain much in Oregon?
- Student:* Because the neighboring states are dry.
- Tutor:* The neighboring states are dry, but, you can not generalize from that. In fact, it does rain a lot in Oregon. Can you guess what causes the rain there?
- Student:* Well, there are high mountains in the region and the states border the Pacific Ocean.
- Tutor:* Yes. The Pacific borders Oregon. How do you think it is involved in the heavy rainfall there?
- Student:* Ocean air rises and cools upon reaching the land.
- Tutor:* That is right.

**Figure 2: The Second Meno-tutor Discourse.**

---

By substituting elementary Pascal looping concepts into the original rainfall knowledge base, Meno-tutor was able to generate a discourse about the looping constructs in Pascal (see Figure 3). This discourse identified some conceptual misconceptions underlying syntactical programmers errors and revealed them to the student. Identification of misconception, in this way, was possible because the knowledge base was built using extensive cognitive studies about how novices learn Pascal constructs [Bonar, 1984; Soloway et al., 1981] and then linking the explicit programming errors with a set of possible misconceptions (see Section 5.2.2).

Two mechanisms are central to discourse planning in Meno-tutor: the tutoring component and meta-rules. The tutoring component is best described as a set of decision-making states organized into three levels that successively refine the actions of

*Tutor:* Do you know that GRADES, in line 9, is a control variable for the WHILE loop?

*Student:* Yes

*Tutor:* Does that mean that GRADES should be

- Initiated before the loop?
- Added into SUM within the loop?
- Modified within the loop?
- Both a) and c)

*Student:* C

*Tutor:* You seem to understand about the control variable of the WHILE loop. Now let's talk about some other command statements. What do you think READ (GRADES) in line 8 does with the value of GRADES?

- Reads all values of GRADES at one time.
- Reads in a single value of GRADES.
- Assigns several new input values to GRADES.

*Student:* A

*Tutor:* Then do you think that  $\text{GRADES} ::= \text{GRADES} + 1$  updates the next of those values?

*Student:* Yes

*Tutor:* There is your misconception. Adding one to a variable is a common way to step through a range of values. But it cannot apply here because your READ instruction in line 8 retrieves only a single value. You should replace line 13 with a second call to READ (GRADES).

**Figure 3: The Pascal Discourse.**

---

the previous level. The three levels, Pedagogic, Strategic, and Tactical make decisions about the form and content of the utterance and each decision is encoded in states at that level. The tutor iterates through the states while running the functions and variable contained in the slots associated with each state. States are linked to each other by a structure which is nominally an OR graph where the nodes are states and the arcs are the default paths between the states. The default path through the OR graph represents context-independent transitions or the tutor's initial plan for structuring the discourse.

The second mechanism is the set of meta-rules that directs the tutor off the default path and onto a path which is ostensibly more in keeping with the particular student or immediate discourse situation. These rules represent the context-dependent

transitions observed in discourse. The path of the tutor can be preempted by meta-rules before entrance to any state.

These mechanisms, as well as the design and implementation of the system, are described in Chapters 3 and 4. Example discourses produced by the system are given in Chapter 5. The implications of this research in the field of education and in the study of discourse processing, along with a statement of my contributions to the field are presented in Chapter 6.

## CHAPTER II

### DESIGN ISSUES IN BUILDING A TEACHING SYSTEM

In this chapter, I examine some of the features and limitations of the tools used to achieve intelligence in earlier teaching systems. I also develop a list of criteria for evaluating the efficacy of these systems. The criteria reflect a "wish list" for building intelligent teaching systems; they are not meant to as a unique definition of the problem of defining intelligence in teaching programs. Rather, they describe a continuum along which solutions to the problem of creating intelligence have been found.

In this chapter, I distinguish between a teaching system and a tutoring system. A teaching system presents new data to a student or provides an environment within which a student can experiment with his information, testing and exploring his reasoning. A tutoring system is a specialized teaching system that engages a student in a linguistic exchange for the purpose of clarifying a body of knowledge to which the student has already been exposed, e.g., through lectures or readings.

Tutoring systems are characterized by an effort to probe a student's level of knowledge. The systems presented in this chapter are teaching system, or components or designs for teaching systems; several are also tutoring systems [Clancey, 1979; Genesereth, 1978; Sevens and Collins, 1977]. The distinction indicates a difference in the motivation and direction of the interaction, rather than in the quality of intelligence of the system.

The first section describes early teaching systems and the eventual introduction of artificial intelligence (A.I.) techniques into their reasoning and decision-making processes. The second section introduces a set of criteria and the systems that will be critiqued. The critique flows through the next four sections which describe the four sections of a teaching system: (1) expert module, (2) student model, (3) tutoring strategies, and (4) communication module. The seventh section provides a summary of the design issues.

#### 2.1 Pre-A.I. Instructional Systems

Teaching systems originally delivered a variety of drill and practice instructions: tasks were presented and then repeated, if the student was found weak in the associated area. The knowledge base in these systems consisted of huge static data bases that incorporated all the facts to be taught. Teaching consisted of a planned excursion through the curriculum. Canned comments or stored tasks were triggered by explicitly anticipated student answers. (See [Atkinsons and Wilson, 1969] for a more complete discussion of these systems.) Such systems were, at best, electronic page turners. They introduced sophisticated graphics (e.g., PLATO [Bitzer, 1961] and TICCET, [MITRE, 1972]), used a variety of mechanisms (e.g., touch-sensitive screens), and taught a variety

of domains (e.g., elementary arithmetic, reading readiness, chemistry, history, and language). Yet, basically, they remained elaborate drill and practice systems unable to recognize anything beyond the expected response. Their repertoire of responses was often rigid, shortsighted, and tedious.

Two things were clearly missing from these early systems: the ability to model the student's knowledge and the ability to react flexibly under a variety of situations. That is, the system was unable to respond differently to different students; it engaged the knowledgeable student in the same way that it talked to the confused one.

Early in the 1970's, teaching systems began to include techniques (such as the use of heuristics) from the newly emerging field of A.I. . Systems which used A.I. techniques (or *Intelligent Computer Aided Instructional* (ICAI) systems, as they became known) differed from pre-A.I. systems in that they distinguished among the components of the teaching system itself and used heuristics to refine the decisions made in each component. As each component became the specific focus of research, the issues central to its reasoning began to emerge, as did the component itself. Once intelligent heuristics were added to the component the system's reasoning powers grew.

## 2.2 Intelligent Computer Aided Instructional Systems

A variety of I.C.A.I. systems are introduced in this section. Each is described later in more detail in connection with the component which best exemplifies the research effort associated with each system. The systems are listed in Figure 4 and an indication of the sections where further descriptions will be found is listed below. GROKKER is the only system that has not been implemented; all others have been implemented and some have been used with students.

BUGGY (see Section 2.5.4) encodes both correct and incorrect processes of simple arithmetic in a procedural network. It can automatically produce 330 "bugs" for subtraction. The system was not built to teach students, rather to demonstrate that errors can be explicit and systematic deviations from correct procedures.

BIP (see Section 2.5.2) tutors elementary programming skills by presenting exercises at a level appropriate to the student's ability to program. The system infers the student's ability by testing him, evaluating the programs he writes, and selecting new exercises appropriate to its model of his success at achieving the prior skill.

GROKKER (see Sections 2.4.2, 2.6.3.) provides an interactive learning environment in which a student can explore the elements of a mathematical theory. The environment consists of mathematical concepts, examples, and results and their logical, conceptual and illustrative relations. It allows the student to display items and to question the system about relevant and conceptually close items. The system defines a

|                 |  |
|-----------------|--|
| BUGGY           | [Brown, J., & Burton, R., 1978]                  |
| BIP             | [Barr, A., Beard, M., & Atkins, R. C., 1976]     |
| GROKKER         | [Rissland (Michener), E., 1977]                  |
| GUIDON          | [Clancey, W., 1979a],                            |
| LMS             | [Sleeman, D. H., & Smith, M. J. 1981]            |
| MACSYMA ADVISOR | [Genesereth, M., 1978]                           |
| SCHOLAR         | [Carbonell, J., 1970]                            |
| SPADE           | [Miller, M., 1982]                               |
| SOPHIE          | [Brown, J. & Bell, A., 1977]                     |
| WEST            | [Burton, R., & Brown, J., 1982]                  |
| WHY             | [Stevens, A., & Collins, A., 1977]               |
| WUMPUS          | [Stansfield, J., Carr, B. & Goldstein, I., 1976] |

Figure 4: Selected Teaching Systems.

metric for further “understanding” a mathematical theory.

GUIDON (see Sections 2.4.2 and 2.7.1) uses a mixed-initiative dialogue and a case-method paradigm to tutor from the information in an expert system. Its goal is to make the knowledge of an expert system accessible to a student. The primary expert system used by GUIDON has been MYCIN [Shortliffe, 1974], a rule-based system which provides consultation for medical personnel about infectious disease diagnosis and therapy. NEOMYCIN is a later, and more successful, version of GUIDON based on making some of the design changes discussed later in this chapter [Clancey W. & Letsinger, R., 1982].

LMS (see Section 2.54) produces a diagnostic model of a student’s errors in algebra. It examines incorrect solution to a student’s algebra solutions and produces a model of his probable misconception. LMS does not do remedial teaching beyond testing and explaining the algebra misconception.

The MACSYMA Advisor (see Section 2.5.4) consults with the user who is interacting with a large and complex mathematical computer system. As the user performs symbolic manipulation of expressions in the system, the Advisor analyzes his input and if the user makes an error, it will generate a plausible “plan” that explains the intention behind his commands. From this plan, the Advisor proposes a misconception that might have caused the user to use the wrong command and then advises him in an utterance tailored to that misconception.

SCHOLAR (see Sections 2.4 and 2.7.1) conducts a sophisticated natural language discourse with a student about the geography of Latin America. It makes a judicious choice of topics, interweaving questioning with presentation, and uses reviews and error correction techniques according to explicit rules of tutoring obtained from an indepth analysis of human tutoring discourse.



SPADE (see Sections 2.4.2 and 2.7.1) provides an environment for the novice LOGO programmer. The environment is designed to teach abstract concepts of programming, such as style, strategies and large-scale organizational techniques. The system allows the student to suggest planning steps and provides a structured way to debug programs.

SOPHIE (see Sections 2.4.4 and 2.7.1) provides a Sophisticated Instructional Environment for electronic troubleshooting and reasons about a student's attempts to debug a simulated electrical circuit. The system simulates probes suggested by the student on the elements of the circuit. It tests the student's conjecture about the cause of the problem and "refuses" to carry out probes that it infers are unimportant to the solution of the problem.

WEST (see Sections 2.4.2, 2.4.4, and 2.6.3) provides a coaching environment for a PLATO game which exercises elementary-level arithmetic skills. The object of the game is to move a player across an electronic gameboard by an amount equal to the value of an algebra expression that the student can write and solve. The coach describes better moves and missing skills that the student might have used to improve his game.

WHY (see Sections 2.4.1 and 2.7.1) is a tutoring system that discusses rainfall and the factors that might be responsible for its production. The complexity of the knowledge in this domain results in a variety of misconceptions that the dialogues are designed to unravel. The program is modeled after human tutoring behavior.

WUMPUS (see Sections 2.5.2 and 2.7.1), the last system discussed in this chapter, is a coaching environment that guides a player through a warren of caves monitoring his attempts to slay a dragon. The player is advised about locating the dragon based on dangers and warnings about bats, pits and the Wumpus itself. The player uses procedural skills to make inferences about multiple dangers in 20 or more randomly connected caves.

### 2.3 The Components of a Teaching System

There are at least four aspects to the process of successful teaching: 1) the ability to work in the subject area, 2) knowledge of the student, 3) knowledge of teaching, and 4) the ability to communicate. Though a teaching system must handle each of these aspects, it need not use a separate module for each. Nevertheless, in this chapter I shall refer to each aspect of teaching as though it had been built into a separate module and shall list the criteria for developing intelligence in that module as though it could act independently of the others. These components reside in an environment that includes language processing modules, as illustrated in Figure 5.

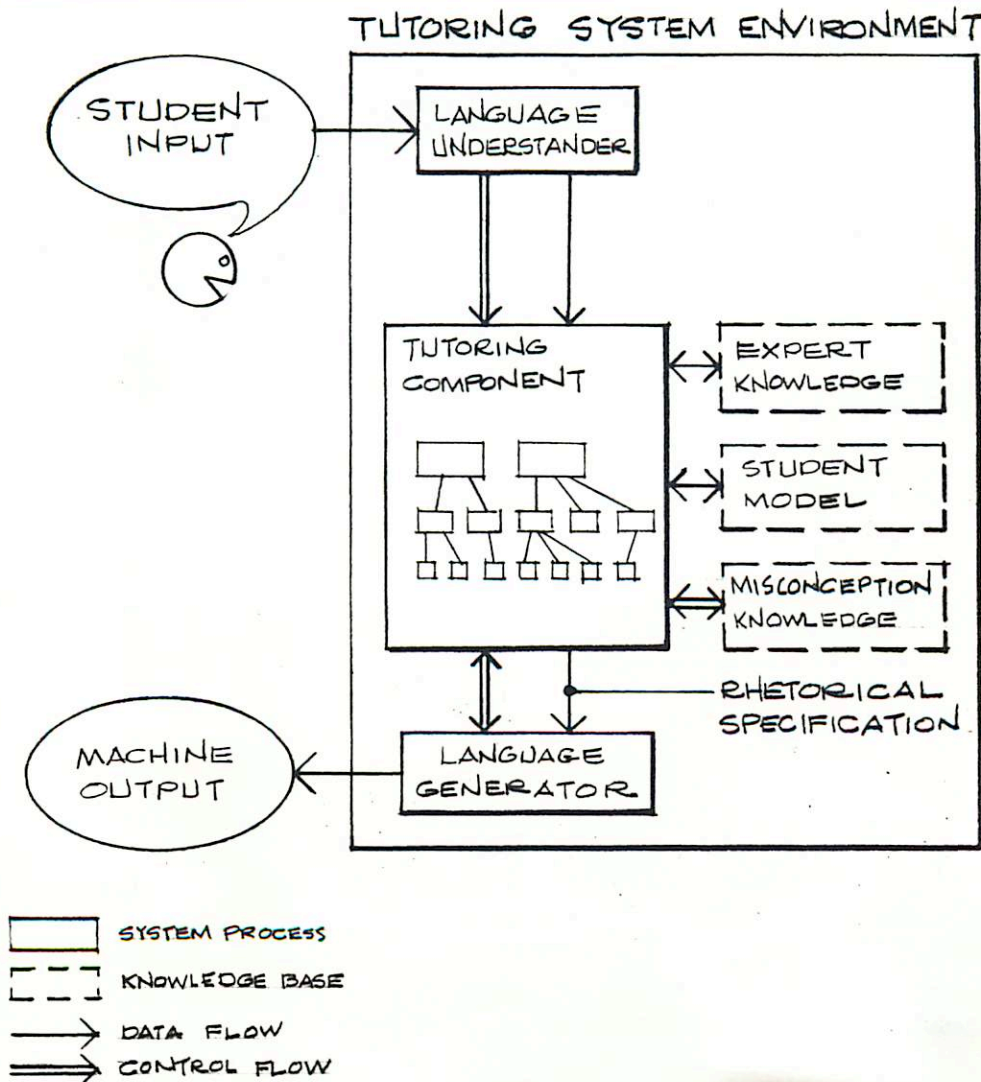


Figure 5: The Environment of a Teaching Machine.

The four components that are the focus of this chapter are listed in Figure 6 and described in detail in Sections 2.4-2.7. The criteria listed in Figure 6 provide a scale or spectrum against which the module's knowledge can be measured. My intention is to judge the completeness and expressiveness of a system in terms of its ability to incorporate intelligence into each module. Evaluating system effectiveness as a function of module intelligence has not, as yet, been achieved. The criteria proposed here provide a beginning to such a systematic evaluation of teaching systems.

The criteria are not meant to suggest that one end of the spectrum holds the "correct" way to include intelligence in a system, or that achieving the criteria will produce a more effective system. I do suggest, however, that linking system effectiveness

**Expert module:** knowledge base of "correct" data and rules from the subject area. The module might be used to present topics for discussion, to perform inferencing on the student's input, or to match the student's answer with the expert's data.

- \* grain size of knowledge
- \* language of the expert knowledge

**Student model:** representation of the student's knowledge, which might contain his history, explicit errors and probable misconceptions. The model is dynamically maintained.

- \* overlay or cognitive model of the student

**Teaching component:** system of rules and procedures to control the teaching activities. For instance, this component includes rules about *how* to teach, *what* tools to try, and *how often* to interrupt the student.

- \* a spectrum between constructivist learning and rote learning

**Communication module:** system of rules to control the input and output behavior of the machine. This module includes rules of discourse, coherency and rhetorical composition (e.g., suggest a topic only if mentioned by the student or logically related to the previous topic and, then, make reference to its earlier use).

- \* rules of discourse and rhetoric

**Figure 6: Criteria for Building Intelligence into a Teaching System.**

with its design criteria can standardize the process of building of designing and building teaching systems.

## **2.4 Expert Module**

The expert module contains knowledge of the subject area and is responsible for evaluating student input against expert data and rules. In some cases it provides problems to solve or topics to discuss; sometimes it is responsible for solving those problems (e.g., SOPHIE). It is the first component of the system that reacts to and must reason about the student's answer. It contains the processes that first translate the student's input into usable form for evaluation against system knowledge.

This component was also the first to be focused on by the A.I. community in their efforts to add intelligence to teaching systems. Once heuristics were built into this module, the system was able to respond to a broader range of input, in some cases understanding and generating natural language (e.g., WHY, SCHOLAR). Adding intelligent heuristics to this component resulted in the system's increased flexibility in understanding the student and its ability to interpret the nontraditional student response, i.e., the response that was couched in words or processes that differed from the norm. It also enhanced the system's ability to express its own knowledge of the domain. Sometimes, as a result of heuristics added here, the system was able to explain itself and talk about its problem-solving reasoning (e.g., GUIDON, SOPHIE). The system was said to be "aware of its own knowledge." This module has been said to contain "domain knowledge" and has also been called the "competence model."

### **2.4.1 Grain Size of Knowledge.**

One of the criteria that either facilitates or limits intelligence in the expert module is the grain size of its knowledge representation. This representation enables the system to make inferences about the student's knowledge and the grain size of that representation determines the "richness" or number of connections between items in the knowledge base.

One metric of this grain size is along an epistemological continuum of knowledge representation beginning with "bits-and-pieces" on one end and ending with "chunked elements" on the other side. At the bits-and-pieces extreme, distinct and unconnected elements are used to represent elements in the subject area (e.g., WHY), whereas at the chunked extreme, relations and morphisms between elements indicate temporal, logical, or pedagogic connections between the elements (e.g., SPADE). The chunked representation uses connectedness or layers of importance and logical precedence of elements to help create grouping for teaching the knowledge.



The grain size measurement indicates more than a passive concern for epistemology or implementation style. It establishes a limit on the flexibility of the system's ability to teach. The primary distinction for teaching purposes is that the bits-and-pieces approach assumes that elements can be taught independent of each other, whereas the chunked approach uses the clustering and richness of domain connections to structure the teaching. The chunked approach, when it is modeled on the abstractions and relations used by a human to learn the subject area, is able to capture the generalizations and strategies of teaching in that domain.

Psychological studies suggest that experts chunk their knowledge in a number of fields: in chess [Chase & Simon, 1981], story understanding [Bower et al., 1979], physics [Larkin et al., 1980], and programming [Gerhart, 1975; Soloway et al., 1983; Bonar, 1984]. These studies suggest that experts use plans, abstractions or templates to understand a domain. Rich computational chunking of mathematical knowledge has been described [Rissland, 1977] (see Section 2.4.2) and Minsky has suggested that the computer's ability to "learn" and "understand" knowledge is ultimately connected to its ability to "chunk" knowledge [1983]. Minsky also examines chunking as it applies to teaching:

I think it's bad psychology when teachers shape our children's mathematics into long, thin, fragile definition tower-chains instead of robust, cross-connected webs. Those chains break at their weakest links, those towers topple at the slightest shove. . . . Perhaps, instead, we ought to help them build more robust networks in their heads. In real life, the best ideas are as cross-connected as can be.

The secret of what something means lies in the ways it connects to all the other things we know. The more such links, the more a thing will mean to us. . . . Rich, multiply-connected networks provide enough different ways to use knowledge that when one way doesn't work, you can try to figure out why. When there are many meanings in a network, you can turn things around in your mind and look at them from different perspectives; when you get stuck, you can try another view. That's what we mean by thinking.

In terms of implementation, the bits-and-pieces philosophy implies, but is not limited to, a sparse number of nodes loosely connected in a semantic network or production rules which are not highly interwoven. For example, GUIDON, a system which was originally a bits-and-pieces system (see next section) was built from hundreds of not too well connected production rules. A chunked system, on the other hand, is often implemented in frames, scripts, or layered and richly connected semantic networks. Most teaching systems lie somewhere between the two extremes.

### 2.4.2 Examples of the Grain Sizes of Teaching Systems.

The WHY system [Bates, 1978; Stevens & Collins, 1977] is an example of a bits-and-pieces representation. (It was written in a semantic network whose nodes were not richly connected.) WHY describes the information about rainfall by explaining the data at the point in the knowledge base where the student's error is stored and then moving to the point where the correct knowledge is located, eventually questioning the student about every intervening topic. WHY tended to badger the student about a myriad of details when it could have inferred the same information from evaluating prior answers or from generalizing over correct or missed topics. By assuming that teaching can begin at any point and can be continued to any connected point, the WHY system attempts to "feed" data to students piecemeal (see later discussion of "wet sponge" teaching module, Section 2.6.4).

The WHY system could have chunked its domain knowledge. In fact, the psychological studies that accompanied the implementation effort uncovered extensive data on how people think about rainfall and how common misconceptions tend to cloud their reasoning. If this data had been included, the system might have predicted student behavior or avoided repetitious questioning by, for instance, exploring the student's knowledge of factors like ocean currents or wind direction and predicting from that his knowledge about rainfall. The system might gloss over, or briefly mention, additional remaining factors and not ask the student to elaborate each one.

When data are connected, common variables can be taught as units, abstracting and generalizing to their common causal factors. This dependency would improve the student's ability to learn in the domain and help to focus the machine dialogue on key concepts.

SPADE [Miller & Goldstein, 1977; Miller, 1982], on the other hand, is an example of a teaching system based on "chunked" knowledge. It uses abstractions and generalizations about programming to discuss the design and coding of LOGO programs. Unfortunately, its chunking is based on the way an expert chunks his knowledge rather than the way a novice would organize his knowledge. It uses terminology and abstractions that the novice could not yet have mastered, making it difficult for him to interact naturally with the program. For instance, Figure 7 illustrates the use of the system to assist a *beginning programmer*.<sup>1</sup> The student is consulting the system about his design of a LOGO program that will draw a "wishing well" -- a square box with a triangle placed on top of a column sitting vertically on top of the box. The terms used to discuss the programming commands refer to planning abstractions that might be used by the expert; they can only be incompletely, if at all, understood by the novice LOGO user.

---

<sup>1</sup> Dialogue from SPADE-0 quoted from Miller, M., "A Structured Planning and Debugging Environment" in *International Journal of Man-Machine Studies*, 11, 1-3, 1979, also in Sleeman & Brown (Eds.), *Intelligent Teaching Systems*, Academic Press, Cambridge, Mass, 1982.

*STUDENT:* Go to the decompose node of Wishing-Well.

*SYSTEM:* The active plan for the decompose of Wishing-Well is Sequential. What now?

*STUDENT:* Edit

*SYSTEM:* Editing the decompose of Wishing-Well.  
Only one choice is left: Decompose -> Repetition. What now?  
A. Iteration  
B. Recursion

*STUDENT:* A

*SYSTEM:* What now?

*STUDENT:* Restore

*SYSTEM:* . . . You were pursuing the first interface of the sequential plan. What now?

**Figure 7: An Example of SPADE-0 Interaction.**

---

GUIDON [Clancey, 1979] is an example of a large teaching system in which the bits-and-pieces approach failed and a later implementation, based on chunked knowledge, succeeded. MYCIN, the large medical expert system from which GUIDON taught, is used by doctors to diagnose and treat infectious diseases. However, doctors work at the so-called "compiled" level where they use rules stripped of the causal reasoning and cross-links needed by a student to learn the same rules. In order to teach from these rules, GUIDON had to "decompile" and cross-index the stripped down rules and provide the student with generalizations and references between the rules.

GUIDON was originally implemented by "reversing" the 450 rules of MYCIN. The original system failed, Clancey said, because medical diagnosis is not made "cook-book" style [Clancey, 1979]. That is, medical practitioners do not diagnose diseases by using perfect recall on a huge number of medical facts and rules. They use common variables revealed, for instance, through visual inspection of the body to suggest potentially new dependencies, for example, "yellow coloring is suggestive of liver disorders." To quote Clancey and Letsinger [1982]:

A psychological model of diagnostic thinking cannot be represented by simply rewriting MYCIN's rules. Instead, the representation and interpreter must be augmented and the rules organized by multiple, orthogonal structures.

For example, a simple interpreter change is to allow incoming data to cause new subgoals to be set up and pursued. When a physician hears that the patient has a stiff neck, the association to meningitis might come to mind, prompting him to determine if the patient has a headache as well. To bring about this effect in NEOMYCIN, a new type of antecedent rule had to be allowed, and a local change made to the . . . control structure.

Clancey's experience suggests that the original GUIDON system was ineffective exactly because connectivity and inferences between elements of the domain were not made by the system. After he added layers of meta-knowledge to include generalizations and common variables on which the system could draw, the system became more effective.

GROKKER teaches mathematical theories using chunked knowledge. It provides examples, concepts, and results as well as relations between subclasses of items to make the logical, conceptual or illustrative association needed to firmly establish the item's meaning in the learner's mind. The student is encouraged to piece knowledge together according to his own needs. He can, for instance, look at analogies or examples from areas that are already familiar to him if that serves to illustrate the subject for him. The model of concepts in a mathematical theory is sufficiently rich and connected to enable a student to work with a variety of views of a single concept.

The student can ask GROKKER to

- Examine related items and counterexamples.
- Watch and interact with graphics.
- Look at items restricted to a set.
- Examine a dual item.
- Look at items that follow from the current concept, example, or result.
- Examine a items that precede the current concept, example, or result.

The important feature of GROKKER is not that a particular dialogue sequence can be created between the student and the system, but rather that the information for multiple dialogues and strategies is available by extracting the appropriate data from the knowledge base. GROKKER's rich view on the data supplies the questions or investigations that should be a part of every student's standard repertoire of inquiry. The questions and the dialogue can be varied by either the student or the Advisor, a separate module that suggests a possible track through the data. The questions and path proposed by the Advisor suggest material needed to fully understand and assimilate the domain.



The rich representation of knowledge used by GROKKER is exemplary of how to build a chunked knowledge base. However, GROKKER does not actively interact with the student nor detect his errors and misconceptions (see Section 2.6.3).

### 2.4.3 The Language of the Expert Module.

A second issue in the design of expert knowledge is the language in which it is encoded. This language had multiple roles. One role is to solve the problem given to the student (e.g., SOPHIE and GUIDON). Another role is to communicate with the student. The first role suggests that the language should be powerful for problem-solving (e.g., predicate calculus or production rules). The second use suggests that the language should incorporate or have access to the student's natural language and possibly a subset of the terms he uses to think about the knowledge e.g., spatial or temporal relations between topics. For instance, if a system solved problems in predicate calculus, it should to represent its solutions in a language other than predicate calculus before it could explain the solution to the student. Most systems fail to do this. Goldstein [1977] referred to opaque experts, systems that cannot explain their reasoning and contrasts them with articulate experts, systems that can explain their problem-solving activities.

The two roles of the language raise some complex issues. Can the same language both problem-solve and communicate to the student? Conversely, could a language that constitutes an expressive language in which to talk to the student have the logical consistency for problem-solving behavior? What is to be done with the rules and concepts that are expressible only in the more efficient problem-solving language and are not directly translatable into the student's natural language? How important is it to make the mechanics of problem solving at least theoretically accessible to the student? For instance, if a system used powerful inference routines to represent expert knowledge should these routines be explainable to the student? How can the inferencing mechanism and the rules be conveyed to the student in terms of his own language? These issues remain active as the development of teaching systems continue.

### 2.4.4 Examples of Languages in the Expert Module.

SOPHIE was a landmark effort in the development of domain languages [Brown & Bell, 1977]. It uses multiple representations of knowledge to reason about electronic circuits and to interact with the student in a natural language. It reasons about the student's input by using syntactically meaningful categories such as "resistors," "transistors," and "measurements," which are associated with a grammar rules to parse the student's input. Each category also specifies the appropriate electronics rule to use, e.g, the Kirkoff Law, to gives the value of the current component given other terminal component values. SOPHIE can answer the student's hypothetical questions about circuit values or can generate explanations about possible faults in the circuit.

SOPHIE has knowledge about the rules it uses to problem solve. It describes this knowledge to the student and reasons about the student's partial solutions. It suggests that a student's hypothetical test on the circuit is superfluous and should not be asked. By carefully translating its reasoning into natural language explanations SOPHIE produced information that might otherwise have been unavailable because of the discrepancy between its problem-solving and the communication languages.

WEST [Burton & Brown, 1978] uses an embellished rule-base language to reason about the student's skill acquisition in a game playing environment. It coaches the student by modelling pieces of it analysis of his skill and expressing these, along with explanations and suggestions about the "optimal" move given the conditions of the game.

SPADE is an example of a system that uses planning phrases to perform problem-solving and to communicate with the student. The generalized LOGO descriptors used as its language for expert knowledge are given to the student to design and write his programs (see Figure 7). The same language of descriptors is used by the sophisticated program understander to check the student's work and to solve the problem of validating his program.

In SPADE, the plans worked well for validating a program, but they could not provide the level of explanation and language idiom used by a novice LOGO student programmer (see Section 2.4). Though the plans were natural for expert analysis they were too novel, too sophisticated, and too academic for untrained LOGO student. They could not be readily incorporated into the student's existing vocabulary.

GUIDON provides an example of the tension produced by the two roles of the language chosen for the expert knowledge. In GUIDON, the language consisting of 400+ rules to diagnose diseases was not an adequate language to use in conversation with the student. The natural language rules were adequate, but but idioms and expressiveness of the language was insufficient. The language was sufficient for problem solving, it enabled the diagnosis and treatment of diseases. However, the student's idiom, as a learner of the rules, required a more complex language, one which included logical relations, generalizations, and associations that the original rule set was missing. This discrepancy led to a reconfiguration of the expert knowledge (see Section 2.4).

## 2.5 Student Model

A good human teacher knows about a student's organization of topics in the subject domain. He can recognize common learning problems and misconceptions and can often distinguish between a student who has correct knowledge but organizes it in an unusual way and one who has an incorrect piece of information. The expert tutor is familiar with the rules and processes used by the student that are different from the expert, though not less valid.

A machine tutor must also have this kind of knowledge about the cognitive processes of the human student. It needs to reason about a student who solves the problem correctly, yet in an unusual way; and it needs to address inconsistencies in the student's view of the knowledge. Knowledge of the student's thinking and strategies resides in the "student model." This data base is the second component in the teaching system to be consulted by the system.

Early systems had almost no student model. At best, they used a stereotypic representation of expert knowledge tagged with topics presumed to be known or unknown by the student. In this section I shall describe several types of student models: *overlay*, *skill*, and *bug modeler*. An overlay modeler is a knowledge base which is a subset of the expert module; a skill modeler is an overlay modeler in which the expert knowledge is represented as skills; and a bug modeler is a knowledge base (not necessarily a skill modeler) in which student bugs and misconceptions are represented.

### 2.5.1 Overlay and Skill Modelers.

The Wumpus student model was designed as an axiomatic base of rules or topics overlaid on the expert knowledge [Carr & Goldstein, 1977]. The student's knowledge was seen as a subset or overlay of expert knowledge; items were tagged as "known," "unknown," or "insufficient data to know." Other systems used the overlay model and represented expert knowledge as skills or a proficiency gained through experience (e.g., BIP), rules (e.g., WEST), or preferred tutoring styles (e.g., WHUMPUS). When skills are used to model expert knowledge, learning is measured in terms of the student's ability to use the skill appropriately (e.g., BIP and WEST) or to explore the skill (e.g., GROKKER). The BUGGY research project (see Section 2.5.3) provides evidence that disputes the validity of using overlay knowledge to represent student knowledge.

### 2.5.2 Examples of Overlay and Skill Modeling.

WUMPUS is a skill modeler, which, in addition, originally used a bits and pieces representation of expert knowledge. Since the coach recognized missing skill, without at the same time recognizing the relative difficulty of that skill it was rather insensitive to the player's ability to comprehend its advice given his present knowledge. Initial attempts were made to develop an evolutionary epistemology which divided the set of rules into phases of increasing skills. The design of the Genetic Graph [Goldstein, 1982] (See Section 2.7.1) provided an indication to the system of the kind of explanation to utter because it represented current skills to prior ones.

BIP [Barr et al., 1976] is an example of a system that uses both an overlay and a skill modeler to represent student knowledge. BIP represents the skills of BASIC programming in nodes of a branching tree, called the "Curriculum Information Network." Each node contains the skill, the exercises needed for testing the achievement of that skill, and correct and incorrect example programs which represent that skill at each

node. In addition, fine-grained knowledge about the evolution of the skill, such as analogies to the skill, generalizations from and specializations of the skill, and relative difficulty of learning the skill are also represented.

The tutor searches the curriculum, the Network, through links that express the relationship between the skills enumerated by the nodes. It uses inferences to move the tutor through the syllabus, making assumptions and evaluations about the student's weaknesses or strategies by evaluating his ability to perform the exercises.

The limitation of BIP lies in the fact that its skill modeler is used as the basis of all its tutoring activities: it can neither understand nor engage in activities beyond those represented on the nodes of its syllabus. It cannot, for instance, reason about a misconception which arises from a student overgeneralizing about a programming concept or attributing power to a construct that it does not have.

WEST is another example of an overlay modeler and it demonstrates how the overlay methodology can break down. WEST was and still is used extensively with students. It coached students who were practicing writing and solving algebra problems. The object of the game is to move a player across an electronic gameboard. The movement of the player is geared to the largest algebraic equation the student could write and solve. The system uses a simple overlay modeler to project the student's activities on top of those that would have been used by an expert in the same situation. The teaching strategy is to discover the student's less-than-optimal moves or missing skills and to describe these to him along with an example of the optimal strategy.

However, a student's actions could often be misunderstood by the system; the designers [Burton & Brown, 1978] noted that students often did not take advantage of features that might improve their performance and for reasons that had nothing to do with ability. For instance, a student could use special features of the system to gain advantages: "bumps" would remove his opponent's "icon" and place it several positions behind his own and "shortcuts" would reduce the normal electronic pathway to the goal. But because the student might enjoy watching the icon move across the board he might not take advantage of these moves. The system evaluated these failures as a lack of knowledge on the student's part and "missing bumps" was incorporated into the student model even though the student deliberately chose not to use bumps. The system recorded that the student was not capable of writing the preferred algebraic equation, when, in fact, the student's failure to do so was predicated on other goals. The original WEST system did not understand this bug, though a later implementation did.

WEST could have either included the bug in its student model or could have questioned the student about his motives before accusing him of not being able to write the requisite equation. However, the system was designed to entertain as well as to educate, and the designers chose not to question the student's goal. Its methodology was to remain unobtrusive and to interrupt as little as possible.

### **2.5.3 Bug Modelers.**

Though the systems described thus far could present a variety of information to a student, they could not deal effectively with his errors. In order to do so, the system would need to include bugs in its student model and would need to bias the tutor's knowledge toward recognizing inconsistencies in the student's reasoning. Several systems exist which are able to recognize a subset of errors (e.g., BUGGY, LMS). Such systems are called bug modelers.

### **2.5.4 Examples of Bug Modeling.**

BUGGY is a system that resulted from an extensive study of student errors in the area of simple arithmetic exercises [Brown and Burton, 1978]. BUGGY represents both correct and incorrect procedures of simple exercises and has developed a methodology for reproducing the errors people make in a procedural skill. For instance, the system can produce 330 "bugs" for subtraction. The correct or incorrect procedures of the same skill are represented and applied to solve subtraction problems. Passage through the procedures results in the application of correct procedures; insertion of incorrect procedures in the network leads to an incorrect solution.

BUGGY was never built to teach students; it was designed and is still used to illustrate the power a system would gain if it could automatically generate all possible incorrect student answers. But the implications of BUGGY are great. It provides computational evidence that student behavior is not a subset, nor a simplification, of expert behavior. Rather, errors are explicit and systematic deviations from correct procedures. As such, BUGGY provides evidence against the use of an overlay model of student knowledge.

The Leeds Modelling System (LMS) uses the student's incremental answers to algebra problems to hypothesize a model of his problem solving activities. The system is designed around a set of correct algebra rules and their corresponding "mal-rules," which are used to predict the incorrect strategies used by the student. Using these rules, the system can accurately predict the observed errors (45% out of a sample group of 15 students).

Other systems are able to recognize the student's exceptional or incorrect behavior. SPADE [Miller, 1982] and the MACSYMA Advisor [Genesereth, 1978] can advise their students about faulty programs or plans though they do not use the faulty programs or plans to explicitly teach the student.

The MACSYMA Advisor [Genesereth, 1978] is significant in that it addresses the problem of how to observe and explain a user's difficulties without actually communicating with him. The Advisor observes the user's interactions with the complex computer system MACSYMA and provides the user with an interpretation that might explain his error. The system parses commands typed by the tutor into the computer and

from these it reconstructs a "plan" that is said to reflect the user's intention or intermediate reasoning. The plan is used to infer the student's underlying belief about how the constructs input to MACSYMA actually work.

MACSYMA Advisor has a simple model of errors: it suggests errors associated with the user's mistaken interpretation of how the construct input to the system works. The Advisor generates a plan about the user's underlying belief based on inference from a simple sequence of commands. It does not, for instance, infer a higher level goal (e.g., solve for two unknowns in this equation) that might have been the student's intention behind the sequence of commands (e.g., divide both sides of the equation by the unknown). The Advisor was implemented for only 3 different problems yet it uses a methodology that seems extensible to other complex and closed interactive computer systems. It does not seem easily extensible to noncomputational domains (e.g., finding bugs in a student's solution of calculus problems).

## **2.6 Teaching Component**

The teaching component contains the strategies, rules, and processes that govern the system's interactions with the student. It holds knowledge about how to teach, and it determines, for instance, what instructional tools (e.g., to provide hints or to ask a question) will be tried, when, and how often. Some of the reasonableness or intrusiveness of the system is determined by the rules in this module.

The tutoring component is the third module to be consulted by the system before responding to the student. The decision made by this module is predicated upon the decision formulated by the expert and student models. It decides what response to make and how to integrate the data from the subject area in a manner consistent with the system's view of the student.

This component is not responsible for language processing, discourse management, or input-output behavior of the system. These activities, required of any interactive discourse system, rightly belong in the fourth component, the communication module which determines the syntactic and rhetorical features of the interaction. The tutoring component handles only how to make statements based on the teaching objectives of the system. It makes decisions about which problems to present, which questions to ask, which hints to provide, and how to further interrogate the student.

Many approaches to this module have been taken. Each reflects a philosophy of human learning and suggests how the student might be affected by system activities. In this section I describe three philosophies of human learning and suggest how they are implemented in terms of rules and strategies in the teaching component.

However, human learning is a complex and as yet incompletely understood phenomenon. It will not be reduced to the simple learning activities presented in this section. I present these caricatures of learning philosophies only in order to qualify the teaching results of the systems discussed in this chapter. That is, the teaching systems employ methods that, implicitly or explicitly, reflect a philosophy of how humans learn. Therefore, I couch the following discussion in terms of philosophies of learning, yet recognize that each philosophy is only a sketch of human learning behavior.

### 2.6.1 Behaviorist Learning.

At one end in the space of teaching methods is rote learning. This philosophy suggests that people learn best by repetitious drill and practice. In this method, for instance, bugs play an inconspicuous role: they are problems which need to be eliminated. Little time is spent understanding or repairing them.

Most pre-A.I. systems explored variations on this approach (e.g., PLATO systems). BIP provides a sophisticated variation -- when the student's answer is wrong, it gives the student a more refined problem that exercises that same skill. The student is expected to learn which skill he used incorrectly by figuring out the intention of the repeated exercise. However, the new exercise or simple statement of the problem does not typically provide enough rich, supporting data to enable a student to weave new knowledge into his existing structure of knowledge.

### 2.6.2 Constructivist Learning.

On the other end of the space of teaching methods lies the constructivist philosophy. This approach suggests that humans learn by assimilating, accommodating and equilibrating their knowledge [Piaget, 1971]. The teaching implication of this epistemology is that the teacher provides a rich "environment" and a wealth of extra information, including alternative views of the topic that the student uses while assimilating and accommodating his knowledge. In order to transform his own thinking and understanding in the domain, the student chooses data and information from the environment that is consistent with his existing structure of knowledge.

In this philosophy, the onus of learning is upon the student. The teacher can facilitate the process by probing weak areas or by clarifying confusing concepts. The Socratic Dialogue, for instance, is a discourse method designed to teach according to the constructivist philosophy [Plato, 1922; Collins, Warnock & Passafiume, 1975]. It uses techniques like overgeneralizing the student's error or applying his results to illogical consequences in order to reveal the error in his reasoning.

Bugs play a central role in this teaching approach. They identify the "site" of an error and make both the site and the nature of the knowledge "around" it explicit. They provide a window into the student's beliefs and a way to begin the learning

process. (See [Sussman, 1973] and [Austin, 1976] for a computational assessment of the importance of bugs to learning.) The teacher cannot know the student's bugs with certainty. Nevertheless, he can try to make them explicit by presenting enough information so that the student can repair and rebuild them, himself.

### 2.6.3 Examples of Constructivist Systems.

WEST attempts to identify the site of errors in the student's game of arithmetic skills and to reveal these to the student. It describes the issues and missing concepts and provides a concrete example of their correct use. The teaching strategy is directed at providing enough information for the student to construct his own knowledge.

The GROKKER system is based on a strong constructivist philosophy. It allows the student to explore examples, concepts and results needed for full understanding of a mathematical theory. The tasks needed for full understanding are enumerated and the system has a "checklist" of items which it anticipates that a student will explore in order to understand a topic. However, exploring an item is not the same as learning that item.

Unfortunately GROKKER does not deal with faulty or inconsistent data. This absence is not a structural problem with the system but could be included in a later version of the system. At this point in its design GROKKER does not attribute bugs or errorneous processes to the misunderstanding of a lemma or an axiom. The system reminds the student of items that have been sources of difficulty in the past, but it cannot interpret the nature of that difficulty.

The view of understanding taken by GROKKER is rather naive. It places a great deal of emphasis on the student exploring items (e.g., watching the graphics, viewing the successor concept) and too little on him interacting with the data (e.g., suggesting variables to test the equation or predict a result). GROKKER's definition of "understanding" includes studying and viewing them. According to the constructivist philosophy, however, understanding must include working the information into one's own knowledge and grappling with accommodation. It should include testing conjectures and validating the results. If the built system were to be used in collaboration with classroom activities, where the student could explore his own knowledge, GROKKER could be used as a reference -- in this case an articulate and encyclopedic knowledge base. However, used as a stand-alone teaching system it could benefit from a component which allows the student to experience the limits of the functions or concepts he is learning. Rissland claims that it encourages independence and spontaneity, yet the independence is based on exploration of the knowledge of an expert and not on exploration of the student's own knowledge.



#### **2.6.4 Wet Sponge Learning.**

The wet sponge philosophy overlaps with the behaviorist learning philosophy. It has two components: (a) the teacher has the requisite knowledge, and (b) the student is both prepared and capable of "absorbing" it in the same form as the expert has structured it. This view of learning implies that teaching includes, and might be limited to, correctly displaying the knowledge in front of the student.

By defining and essentially teaching from the knowledge of the expert, a teaching system implies that learning consists of copying the expert. In fact, thinking and reasoning like an expert is probably necessary, but not sufficient, for learning. Psychological studies in a variety of fields suggest that experts and novices have distinct knowledge of the same subject area (see, for instance, [Chase & Simon, 1981; Larkin et al, 1980; Soloway et al., 1983]). The attributes that distinguish a student's reasoning from that of the expert are exactly those attributes that must be taught by a tutor or a system. The variations in the way the novice structures his knowledge as compared to that of the expert must be addressed, and a machine tutor should be equipped to handle this difference.

### **2.7 Communication Module**

The communications module is the last component of an intelligent teaching system to be consulted about the system's response to the student. Historically, this component has been the last to benefit from sophisticated A.I. techniques; today it reflects the least application of heuristics and techniques for managing the dialogue between human and system.

A communications module is a necessity for any discourse system: it monitors the conversation between human and system, sometimes allowing either to initiate it. It must unambiguously receive the human's input. Because a student – even more than a typical computer user – cannot express what he does not know, a teaching system – even more than the average discourse system – must be equipped to recognize deficiencies in the student's articulation of his problem. The system must analyze not only the actual words, but the unexpressed beliefs and processes behind it. This can be inferred from what the student says and from gaps in his knowledge.

The primary activity of the communications module is to converse with the student. I do not mean simply to handle natural language processing – many systems can both parse natural language input and generate natural language output. A system with language processing capabilities does not necessarily interpret the student's meaning or evaluate his knowledge using a rich understanding of novice problem solving strategies.

By communication, I mean "understand the student's deeper meaning." The problem of understanding the student becomes acute when the student talks about knowledge that is organized in a way different from the stored expert representation. In this case can the system know what the student is talking about? Can it use inference techniques from other modules to interpret the student's intent or state of knowledge? Understanding a user requires the ability to ask the right question and to focus on the correct issues.

The communication module, for instance, is responsible for generating *mixed initiative* dialogue that allows either the student or the system to question the other about local concepts. This kind of interaction is responsive on a local level; the student is allowed to ask the questions and eventually the system takes control and resumes the main topic. Production rules are often used in this component to allow the student to pursue some subgoal while enabling the system to regain global control after a fixed number of interactions, or when the student relinquishes control [GUIDON, SPADE].

### 2.7.1 Examples of Communication Modules.

SPADE allows locally mixed initiative and has a strong communications model. It can understand the student's input commands and questions (both represented as single key words) as well as the resulting LOGO program. However, because the student model is very weak and its expert model is based only on "planning skills," it is not able to interpret student commands that are not equivalent to the expert's approach in the field.

WHY has a strong communication model and accepts natural language input. However, because the student model is so weak, it can only mildly understand the student's intentions and cannot use the communications module to focus on a specific topic within a particular subgoal.

The Genetic Graph (GG) [Goldstein, 1982] is a proposed modification to the original WUMPUS coach which, because it represents relations between the skills of the game, would guide the coach to generate more sophisticated utterances about the skills that a student should be focus on. The GG encodes generalizations, analogies, deviations, and simplifications of each skill in a modified semantic network and guides the coach through these skills and relations between skills. First the GG suggests which skills to discuss, namely those on the frontier of the player's knowledge. Then it supplies to the coach about expressing that skill in a natural language utterance, e.g., as an analogous instance of a previously learned skill ("Oh, Mary, you remember we had the same situation when you were in Cave 15. . .") or as a generalization of an earlier skill ("Mary, since you have two warnings about Cave 15, you can infer that it is more dangerous to enter Cave 15 than to enter another one with only a single warning. . ."). The GG provides the coach with knowledge about how to discuss each skill and provides the tutor with "insight" about which skills are premature to discuss given the

player's knowledge as represented in the GG. (See Section 3.3.2 for a discussion of an my first-pass implementation of the GG).

SOPHIE has a strong communications model: natural language capabilities based on a semantic grammar that copes with linguistic properties such as anaphoric references and ellipses. SCHOLAR also provides natural language processing based on a case grammar system which enables it to construct questions and to present new material from the semantic grammar.

GUIDON has the most sophisticated communication and dialogue model. It includes knowledge of discourse patterns and the means for varying the strategies that the tutor uses to guide the dialogue. Tutoring rules are used to decide when a remark might be appropriate, whether to take the option and what to say. One rule, for instance, uses "entrapment" to force the student to make subsequent answers that will reveal some aspect of his understanding (or misunderstanding.) Other rules explicitly define how a subgoal will be discussed, e.g., suggest a subgoal, then discuss the goal, and finally, wrapup the discussion of the rule being considered. The system is able to carry on a flexible dialogue by switching the discourse at any time to portions of an AND/OR tree, which represents the domain knowledge.

## 2.8 Summary

This chapter presented a framework for designing intelligence into an A.I. teaching system. It described how A.I. techniques were introduced into early teaching systems and provided some criteria for evaluating the effectiveness of the techniques in each. Twelve teaching systems were discussed by their ability to build intelligence into the components of the system. Four components were defined to correspond with the four aspects of good teaching: 1) expert module contains a representation of the subject area and is responsible for evaluating student input against expert data and knowledge; 2) student model contains knowledge about the student's organization of topics and the reasoning processes that he seems to be using; 3) teaching component contains strategies that govern the system's teaching interactions with the student; and 4) communications module is responsible for couching the utterance in rhetorically sound phrases focussed on the appropriate issues.

## CHAPTER III

### DISCOURSE ANALYSIS AND THE DESIGN OF A COMPUTER TUTOR

1

Complex knowledge and strategies are required to produce high quality and natural sounding discourse. This knowledge goes beyond the use of a rich grammar or even an adroit planning component. It requires expert communication skills, knowledge of the domain, and knowledge of the audience's level of understanding of the domain. In this chapter, I examine the theoretical and practical issues involved in generating discourse and describe the mechanisms used in Meno-tutor to formalize the knowledge and strategies required to produce reasonable discourse.

The first section discusses the theoretical issues involved in the design of a system for discourse including a taxonomy of discourse based on an analysis of human discourse protocols, an outline of complex speech acts used in discourse, and a discussion of the need for discourse management.

#### 3.1 Understanding Discourse

Intelligent behavior has been modelled by many Artificial Intelligence (A.I.) systems. In these systems, the behavior modelled can be characterized as having explicit goals and rules. For instance, in game playing and theorem proving the desired result is already known and specified; e.g., in chess the goal is winning and winning positions (checkmates) can be defined precisely. In such areas, the research effort has been to work on the heuristics, i.e., to develop ones that are more efficient or more robust. However, many other forms of intelligent behavior are not so well-defined. Carrying on a conversation with a person or writing poetry are examples of intelligent behavior where the rules are not explicit. In these areas a major effort is required to even characterize the results toward which the heuristics are supposed to guide the system [Haugeland, 1982].

In discourse, factors other than whether the statements are true are important. For instance, communication in a discourse goes beyond the text, i.e., the effect of the speaker's words on the rest of the participants is difficult to assess. Each participant makes decisions or interpretations about the spoken words and his interpretations about these words influences his utterances. The use of inferences and interpretation in

---

<sup>1</sup> Portions of this chapter appeared in the Proceedings of The Computer Human Interactions Conference, 1983 sponsored by ACM SIGCHI, co-authored by David D. McDonald.

discourse suggests that a speaker can propose a direction for discourse, but cannot force that direction. As Drew McDermott says, "The speaker does not quite know the organization of the hearer's mind, [and so] his phrasing of information and packaging must . . . be an artwork of suggestiveness and insight" [McDermott, 1982, pg. 151].

This consideration leads to the possibility that *discourse opens new possibilities*. New topics often extend the listener's thinking in the domain and cause him to consider information outside the topics he knows about or those that have already been discussed. The listener's reaction to the new topics often gives the speaker new information about the participant's knowledge and about the intersection of their disjoint models of the domain.

The theoretical issues discussed in the above section should not discourage us in our attempt to build discourse systems. In fact, research on natural language generation has reached a level of technical competence where the proper goal is to try to match actual human performance in discourse with machine generation systems. The practical aspects of machine discourse suggest that we might learn the strategies, high level shifts, and speech patterns in human discourses and use these to build discourse systems. To this end, my work on tutoring has included an examination of human dialogues and discourse management issues. These dialogues, recorded in earlier A.I. work, will be described in this section.

I analyzed portions of twelve human dialogues [Collins & Stevens, 1980; Collins et al., 1975; Grosz, 1976; Pollack, 1982] and arrived at a four-part taxonomy (see Figure 8) based on differences in the speaker's goals and in the assumptions he apparently made about the listener. My goal was to recognize speech patterns used by the speaker and to correlate these with what I inferred to be the intention of the speaker or the assumptions made by him about the listener.

The most potent distinction for my purposes is between *guidance discourse* and *reconstruction discourse*. In *guidance discourse* the speaker's goal is to accomplish a task; the speaker directs the listener to largely new information without concern for what he already knows. On the other hand, in *reconstruction discourse* the goal is to clarify and correct the listener's faulty or confused knowledge, for instance, by pinpointing misconceptions through careful diagnosis and argumentation.

An instance of *guidance discourse* is cited by Grosz [1976], who analyzed a protocol of a speaker guiding a novice through the first steps of assembling a water pump. In this protocol, the speaker directs and corrects the novice but largely ignores his prior knowledge. He neither interrogates nor attempts to change the novice's beliefs about how a water pump works.

Another protocol, analyzed by Pollack et al. [1982], is in the format of *reconstruction discourse*. It is a dialogue between a tax expert and a lay person about how to claim religious deductions on one's tax return. During the discourse, the expert convinced the woman to adjust her ideas about how to record a charitable deduction

**GUIDANCE DISCOURSE**

| <b>SPEAKER'S GOAL</b> | <b>ASSUMPTIONS about the LISTENER</b> | <b>SPEECH PATTERNS</b>   |
|-----------------------|---------------------------------------|--|
| Accomplish a task     | Minimal experience                    | Provide instructions and correct the listener's words or actions |
| Explore knowledge     | Incomplete knowledge                  | Provide instructions and interrogate                             |

**RECONSTRUCTION DISCOURSE**

| <b>SPEAKER'S GOAL</b>                  | <b>ASSUMPTIONS about the LISTENER</b> | <b>SPEECH PATTERNS</b>                     |
|--|---------------------------------------|--|
| Rebuild the listener's cognitive model | Confused knowledge                    | Interrogate and correct                    |
| Change the listener's assumptions      | Wrong knowledge                       | Interrogate and provide reasoned arguments |

**Figure 8: Differences in the Speaker's Goal.**

and caused her to accept a solution previously rejected by her for the wrong reasons. In the dialogue the expert tested the novice's almost complete knowledge and "remade" several imperfect areas.

*Reconstruction discourse* is among the most tightly coherent found in speech because of the amount of knowledge that contributes to it. The expert speaker in this kind of dialogue must know about:

1. typical "correct" solutions,
2. errors and misconceptions which may suggest the need for reconstruction,

3. differences between expert and novice models of the knowledge, and
4. strategies which can bring an incorrect model around to a more acceptable form.

Working with this much knowledge can effectively direct the selection of content.

### 3.1.1 Complex Speech Acts.

My analysis of protocols of human discourse presents evidence that the speaker's intentions determine, to some extent, the speech patterns used. One of the striking commonalities in the twelve human dialogues I studied was the presence of the same patterns of speech across sessions and across speakers. These patterns, along with "packages" of phrases, were used as templates to handle specific types of audiences under specific conditions.

For instance, the "grain of truth" pattern illustrated in Figure 9 (excerpts from [Stevens et al., 1982]) was used primarily when the student's answer was *incorrect* yet when it indicated a partially *correct* understanding of the problem. The tutor's response in both instances was to pick out the "grain of truth" in the student's answer, to acknowledge it, and then to propose the correct answer.<sup>2</sup> The tutor's response was "packaged" in a marked syntactic pattern; the reinforcement was kept short and always appeared in the same position in what was effectively a template.<sup>3</sup>

These discourse segments illustrate the use of a single template response to handle topics of varying complexity. In the first discourse, the tutor deemed it possible and appropriate to state the correct answer: "[warm air] rises [as] it cools off." However, the tutor in the second segment possibly decided that the topic "wind over water," was too complex or too important to simply provide the answer and move on. Instead, that tutor chose to pursue the next logical causal factor by asking a leading question about the effect of a large cold body on rainfall. His question manifests his "hidden agenda" which seemed to be to explore a commonly held misconception as soon as it is suggested in the discourse. In these two discourses, the misconception that was explored was that rainfall is caused by large land or air bodies.

---

<sup>2</sup> Though the "no" of utterance 2 is missing from the second discourse, I believe it is because the student in that discourse indicated uncertainty about the answer.

<sup>3</sup> There *are* variations in the two discourses. Utterance 5, which supplies the answer in the first discourse, has been put off in the second. A "leading question" (Utterance 6) is presented in both discourses to give the student a clue as to how to deduce the answer for himself. I assert that these two passages reflect the same speech patterns because I believe that their similarity in form and tempo overrides their "tactical" variation in content.

**DISCOURSE 1:**

**Tutor:** What happens to the temperature of moist air when it rises?

**Student:** It gets warm.

1: Pose a question.

Note: student's answer is wrong.

**Tutor:** No, warm air [does] rise.  
But, as it rises it cools off.

2: Indicate wrong answer.

3: Reinforce "grain of truth."

4: Indicate contrast.

5: State correct answer.

**Tutor:** Do you think the mountains in  
Washington and Oregon have anything  
to do with cooling the moist air?

6: Ask a leading question.

**Student:** Yes.

---

**DISCOURSE 2:**

**Tutor:** Can you tell me what causes air to rise over Ireland?

1: Pose a question.

Note: student's answer is wrong.

**Student:** The wind?

**Tutor:** The wind moves over the air.  
But, that is not what causes it to rise.

3: Reinforce "grain of truth."

4: Indicate contrast.

**Tutor:** Do you think the cold  
air mass that is usually sitting  
over Ireland might be important?

6: Ask a leading question.

**Student:** If it just sits there it is.

---

Figure 9: Common Speech Patterns and Strategies in Discourse.



### **3.1.2 Discourse Management.**

Managing a machine discourse is both time-consuming and nontrivial; it goes beyond merely formulating explanations or proposing questions. Practical considerations include how to make decisions and why to choose topics that move the user towards an appropriate goal. Like any good speaker or writer, the system must be attentive to its audience. It must keep the listener's attention while maintainin a relevant and coherent discourse; it should at the same time be sensitive to the listener's failings and apparent confusions. This section describes the need for discourse management and the methods chosen to achieve it in my research.

#### **Tailoring the Discourse.**

Observations of human tutoring protocols [Stevens et al., 1982] (see Section 3.1) and extensive interviews with novice students in several domains (see for instance, [Collins et al., 1975] and [Bonar, 1982]) have revealed patterns of tutoring which must be incorporated into the knowledge of a tutoring system. Psychological studies [e.g., Collins et al., 1975] describe tutoring as a process where the discourse is tailored to the specific needs of the student, and one which is designed to probe a student's strengths and weaknesses. In successful tutoring, the tutor expands on the student's knowledge and presents new material only when its relation to the established material is clear, e.g., when the new topic is keyed off of a previous topic in the discourse.

The effective human tutor does not typically follow a fixed list of topics nor a fixed pedagogical approach. Rather, both are produced on the fly and in response to the tutor's perception of the student's knowledge. Tailoring a discourse to a student means having the ability to

- focus on appropriate topics,
- present logically related data, and
- select heuristics for shifting topic and tutoring style.

Meno-tutor's particular solution to tailoring a discourse is presented informally in the next section and in more detail in Chapter 4.

#### **Relevance in Discourse.**

In addition to being able to tailor the discourse, the effective tutor is apparently able to focus on relevant topics. Focusing on relevant topics, for a machine, means using a robust model of the student and not presuming, for instance, that the student understands more than he does, or alternatively, that he understands less than he does. A discourse system should converse only about topics which can be understood by the

student and only about those that he has some interest in discussing. It should not, for instance, dwell on topics that are trivial or beyond the student's ability to understand.

In computational terms, relevance means that the system knows not only *what* to talk about, but also

1. *how* to update the model of student knowledge,
2. *what* additional knowledge should be known in order to learn the new data, and
3. *which* presentation method is most effective.

#### Recognizing Unspoken Intentions.

One of the basic issues to be addressed in tutoring is how to recognize a student's unspoken problems or intentions. We call this the "tip of the iceberg" problem, by which we mean that the words spoken by the student or the teacher are only a small portion of what they know about each other and what they are communicating in the dialogue. This problem has been analyzed by Allen, who built a system to model the knowledge needed by an effective agent at a train station [Allen, 1979]. The discourse in Figure 10 is in the style of Allen's examples and is predicated on the agent anticipating or predicting the traveler's problem.

---

**Traveler:** "Is this the gate for the train to New York?"

**Agent:** "No, you want gate number 44 and the train leaves at 6:33."

**Figure 10: Discourse to Handle Unspoken Problems.**

---

If the answer to the traveler's question had been "yes" that by itself would have been enough and would have solved the traveler's problem. However, since the answer was "no", a simple "no" by itself *would not* be a sufficient answer because it would not have addressed the traveler's actual problem. By giving the additional information, the agent showed that he recognized that what the traveler was *saying* was only a small part -- only the tip of the iceberg -- of what he actually intended to communicate.

The "tip of the iceberg problem" in tutoring is to recognize the student's unspoken confusions and misconceptions. By maintaining a record of the student's previous errors or by directly questioning him about his misconceptions, a machine tutor can acquire the same kind of information as used above by the train agent above.

A human tutor uses the discourse process to recognize unspoken confusions or inconsistencies in his student's knowledge. Similarly, a machine tutor, must use the same discourse process to recognize unspoken problems. During the normal course of machine/student interaction, a great deal of information, particularly linguistic information, can be recovered and analyzed about (1) previous errors, (2) the depth of the student's knowledge, and (3) his possible mistaken goals. As I demonstrate in the research, a machine can use interactive conversational tactics to recover such linguistic information.

### The Role of Errors in Discourse Management.

Errors are powerful clues in the hands of a tutor; they provide an opportunity to explore and track a "site" of confusion in the student's knowledge. Focusing on errors allows the discourse to be used as a vehicle to reveal and repair false beliefs. In fact, identifying and correcting errors sometimes provides an opportune learning strategy – if the student does it right the first time, he already knew how to do it and so was not taught anything.

The importance of errors to learning has been validated in early A.I. learning systems. For instance, Austin [1976] presents a computational view of juggling where bugs in the learner's motor skill procedures are clearly explained in an effort to help him refine and improve his learning skills in juggling. Sussman [1973] developed a computational model of skill acquisition in which the machine learned to write computer procedures, in part because it could refer to its library of erroneous procedures, which it tailored and remade to fit future problem applications. In another learning system, Winston [1975] showed that the definition of a concept could be learned by a machine by presenting it with examples of that concept. The system used errorful examples, or "near-misses," to cause structural changes in its definition of the concept. For instance, while learning about an "arch," the system was shown a "near miss" example in which columns were touching. Viewing this example initiated a refinement to the system's definitions of arch which would include "a structure whose columns do not touch". When the system was presented with a true or correct example its initial configuration remained intact.

A student's learning behavior might be similar to that of a machine: when the student makes an error he must adjust his conceptual model and rebuild his knowledge to accommodate the changes dictated by the error. However, when his original answer is correct, he does not make modifications to his original concept, though he may extend or generalize that model.

This is not to suggest that other teaching methods are not important; they certainly are. Reinforcement learning, for instance, encourages continuation on the current path; validation learning reinforces existing knowledge. Each learning strategy is effective in different situations.

### 3.1.3 Alternative Dialogues.

There are several ways for a discourse to proceed and several ways for a tutor to interact with a student. Each choice produces a different subdialogue as well as a potential new direction for the discourse. This section discusses alternative dialogues and the management issues involved.

Clancey [1979] discussed alternative discourses and represented them in terms of a state transition diagram. In this research, I extend the concept to include decision points dictated by pedagogical, strategic, or tactical considerations. The *Discourse Management Network* (DMN), Figure 11, links changes in the control structure of the tutor to the planning of discourse text for the dialogue of Figure 1 reproduced here in Figure 12.

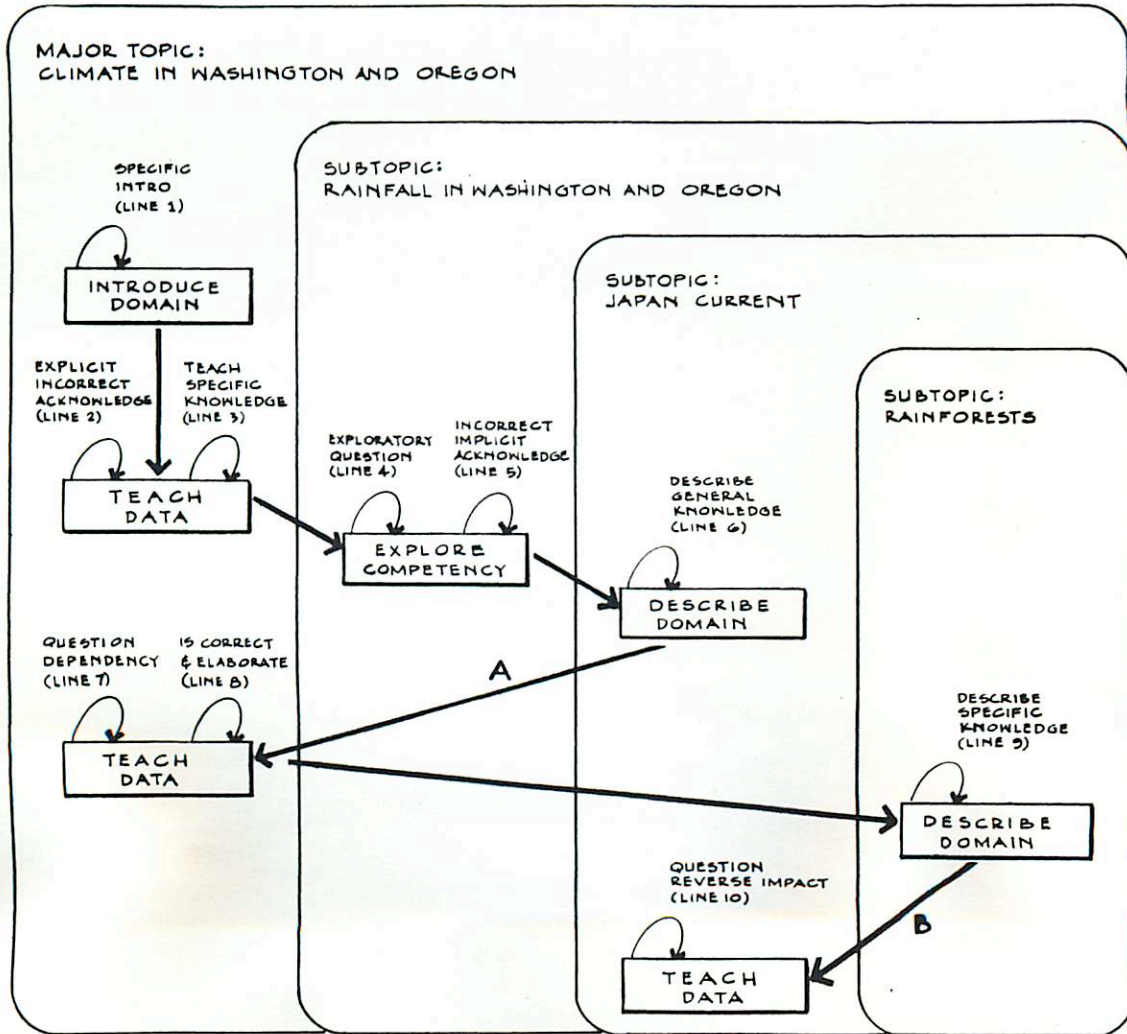
Meno-tutor selects from a variety of "virtual utterances," planning and checking the environment before formulating its text. Each refinement is specified in a general and schematic way. Alternative branches in the DMN indicate options as to both topic and style. *Whether* to take a particular option (i.e., whether it is opportune to follow a particular approach) and how to determine *what* to talk about (i.e., what is the most salient topic) are determined dynamically by the control structure (see Chapter 4).

Throughout the trace, alternative multisentential texts are available, though not shown in the diagram. The DMN represents a decision made, a tutorial style adopted, and a topic chosen for each component of the tutor's text. For example, in the first five statements, the tutor introduces the topic, corrects the student, and continues to explore the student's frontier of knowledge. By line 6 the tutor reaches the point of considering primary data to build up the student's knowledge of the causes of rainfall. In order to do this, the tutor shifts from statements about general weather descriptors to a description about the Japan Current. There are two reasons behind this shift: (1) the student lacks sufficient knowledge about the basic concepts as evidenced by two wrong answers, and (2) the data base contains supplemental knowledge about rainfall upon which the climate of Washington and Oregon is predicated – the Japan Current. Because this supplemental knowledge is a factor in creating rainfall, the tutor may have assumed that the additional data would enable the student to deduce the correct answer.

After presentation of relevant information about the Japan Current the tutor asks the student (line 7) to identify the effect of the current on the rainfall in Washington and Oregon. In doing so, the tutor explicitly returns to the major topic of the discourse, "climate," both to establish the connection of the Japan Current to it and to return to the major focus of the discourse. This transition is marked with an "A" on the DMN.

The tutor's next utterance after an obligatory acknowledgement of the student's correct answer is to elaborate about the effect of the Japan Current in a description of the rain forests (line 9). Rain forests are causally linked to warm and rainy climates, and so the tutor may have provided the additional information to supplement the student's growing information about weather in that area. In line 10 (marked with a "B")

Figure 11: Discourse Management Diagram (DMN).



- Tutor:* 1. Do you know what the climate is like in Washington and Oregon?
- Student:* Is it cold?
- Tutor:* 2. No,  
3. it is rather mild.  
4. Can you guess about the rainfall there?
- Student:* Normal, I guess.
- Tutor:* 5. Well,  
6. the Japan current, which starts in south eastern Pacific, goes along the coast of Japan and across the North Pacific ending up off the coast of Washington and Oregon.  
7. How do you think that current affects the climate there?
- Student:* It's probably rainy.
- Tutor:* 8. It is indeed very wet,  
9. there are rain forests in both Washington and Oregon.  
10. What does that tell you about the temperature of the Japan current?
- Student:* It's warm.

---

**Figure 12: The Rain Dialogue.**

---

on the DMN) the tutor returns to the topic of the Japan Current and asks the student to make a reverse inference about an aspect of the Japan Current (temperature) based on the existence of rain forests in Washington and Oregon.

### 3.2 Design of the Computer Tutor

The theoretical issues discussed in the previous section suggest that the design and implementation of a computer tutor involves complex control structures which can arbitrate among multiple knowledge. To construct such a computer tutor we need to integrate large bodies of knowledge in complex ways. In Meno-tutor, I implemented several mechanisms which formalize some of the concepts of planning and reasoning listed above. Three mechanisms are discussed in this section: levels of planning knowledge, tutoring states, and an annotated knowledge base. The control structure that engineers the interaction of these mechanisms is described in detail in Chapter 4.

The first mechanism discussed is the levels of planning knowledge used to make decisions about the approach and content of the machine utterance. Rather than articulate an exact form for the plan of discourse text, the levels of planning allow the system to study the ways that discourse can be structured and to evaluate eventual output. The levels of planning are important for allowing the tutor to structure the available alternatives of discourse text and tutoring approach.

The second mechanism is the representation of tutoring decisions in the form of states. States contain procedures which execute routines that can call on other pieces of knowledge from the other sources. The states evaluate registers which formulate the student's knowledge or the state of the discourse to see whether the approach of the discourse makes any sense. This is particularly important in handling discourse, where the interpretation of a student's answer may depend in complex ways on the preceding discourse and the knowledge of the subject matter.

The third mechanism is the annotated expert knowledge base, which records the pedagogical salience of topics in the subject area along with the system's view of the student's grasp of each topic. This designation is used to provide both relevance and pedagogical alternatives in the system's introduction of new topics.

These mechanisms cannot be viewed independent of the control structure which arbitrates among them and chooses data from within them. Yet they are discussed separately here and the control structure which arbitrates between them is discussed in detail in the next chapter.

### 3.2.1 Levels of Planning.

Discourse planning in Meno-tutor is accomplished in three planning levels. Each level refines the plan made at the previous level and further constrains the form and content of the output utterance. The levels are:

- *The Pedagogic Level:* a choice is made about the style of tutoring, for instance Socratic or coach-like tutoring. The choice is made before the discourse begins and is typically not reconsidered unless the student performs poorly.
- *The Strategic Level:* a choice is made between teaching strategies which implement the pedagogy chosen above. The choice is made from among highly schematic scripts that determine an expository style and activity, such as "question the student," "describe the concept," or "choose a new topic."
- *The Tactical Level:* a choice is made between available speech patterns and language structures to implement the strategy from above. At this level the conceptual content is chosen to fit the script and topic specific data is considered for the first time.

The functionality of each level is considered below and the use of states in each level is discussed in the next section.

### **Pedagogic Level.**

The Pedagogic Level establishes an expository style for the dialogue. The choice determines, for instance, the number of interruptions or the amount of probing. For instance, a coach-like system might interrupt the student as little as possible; a Socratic-like discourse might interrupt frequently, directing the student to recognize logical inconsistencies of his reasoning. I will not say more about this level of the planner as my focus for the rest of the thesis is on the next two levels of the planner.

### **Strategic Level.**

Strategic Level is the kind of knowledge that makes an expert tutor; it is the discourse knowledge people refer to when they talk about an excellent tutor. Strategic planning allows the tutor to *focus* on a topic or a strategy for the purpose of explanation or testing. It determines, for instance, which expository style to use and how to smooth the discourse between expository styles. At this level, a choice is made between, for instance, using in-depth probing or providing lengthy descriptions, explanations or analogies to the student. For instance, the schematic scripts at work in the examples on page 38 were “probe for a misconception” and “repair the misconception.” A refinement to the strategy of choice in these dialogues was made in the second dialogue when the tutor used “repair the misconception” without first providing the correct answer.

Strategic planning determines the underlying number and kinds of interactions used by the tutor. It generalizes the tutor’s approach to allow for an interplay between approaches such as complete the topic, and introduce a new topic. Thus, it permits statements to flow into each other.

### **Tactical Level.**

Knowledge at the Tactical Level is used to implement the strategic script in a specific domain and according to a tactical refinement of that script. It uses context-dependent topics and to “flesh out” the context-independent strategy described above. For example, at the Strategic Level a choice might be made to ask a question; at the Tactical Level that question will be formatted into a question about a specific element in the expert knowledge base, or into a question about the relation between a role of that element and, perhaps, an element discussed earlier.



The knowledge in the Tactical Level includes task-specific techniques for constructing the utterances. For example, a tutor of geography knows how to question a student about causal relations between the development of a city and the economics or climate of the region. A tutor of programming languages, on the other hand, has scant need to talk about causal relations. Rather, he couches his questions in terms of values of variables or the descriptions of algorithms. The tactical knowledge used in each domain is different. For instance, independent factors in geography (e.g., wind direction and coastal currents) *cause* events to occur (e.g., rainfall and the development of rain forests); independent factors in programming (e.g., variables, value bindings, boolean relations) *determine* activities (e.g., loop termination, conditional branching). The wording of the relationship between the factors needs to be distinct in the output text.

A skilled tutor uses tactical knowledge to add cohesiveness and completeness to the dialogue. For instance, it discusses the relation between a current topic and prior ones before moving to new topics. Or it uses the "grain of truth" technique demonstrated above to relate a specific correct element in the student's information to a correct formulation of the same topic. Another way to "bridge" the gap to a new topic is to elaborate upon previously mentioned topics which may impact upon the new topic. Tactical rules, for instance, suggest that the tutor discuss the influence of subtopics before moving on to new topics.

### 3.2.2 Representing Discourse by States.

In this section I describe the states used to implement each of the three planning levels. States constrain the form and content of the utterance and hold the elements of style and content used to realize the utterance. States are organized into a tree whose leaves fall at three different levels. Figure 13 suggests the functional states and the decisions available at each level. The tree contains 40 states each built from a Lisp structure that holds slots containing variables or functions to be evaluated once the state is entered. The code for all 40 states is given in Appendix B.

The particular form taken by an utterance within the discourse is determined by the state in which the tutor finds itself at the time of the utterance. For example, when line 6 in Figure is printed the tutor is in state *describe specific topic*. Then the tutor moves into the state *teach* where the tactical plan *question reverse impact* (line 7) is executed. In response to the student's first correct answer after two wrong answers, control moves to the state *is correct and elaborate* (line 8).

The state *acknowledge with elaboration* is entered in line 8, instead of the more simple *explicit acknowledge*, as was done in line 2 above, because the context is different. In the second response, a register is set when the knowledge base is scanned, indicating that roles and attributes of the current topic, "Japan Current," are linked to topics in the domain which remain to be discussed. One of these related topics, "rain forests," is an extension of "Japan Current" and is associated with "rainfall in Washington and Oregon." It is chosen as the topic of the next textual output, and the state *acknowledge*

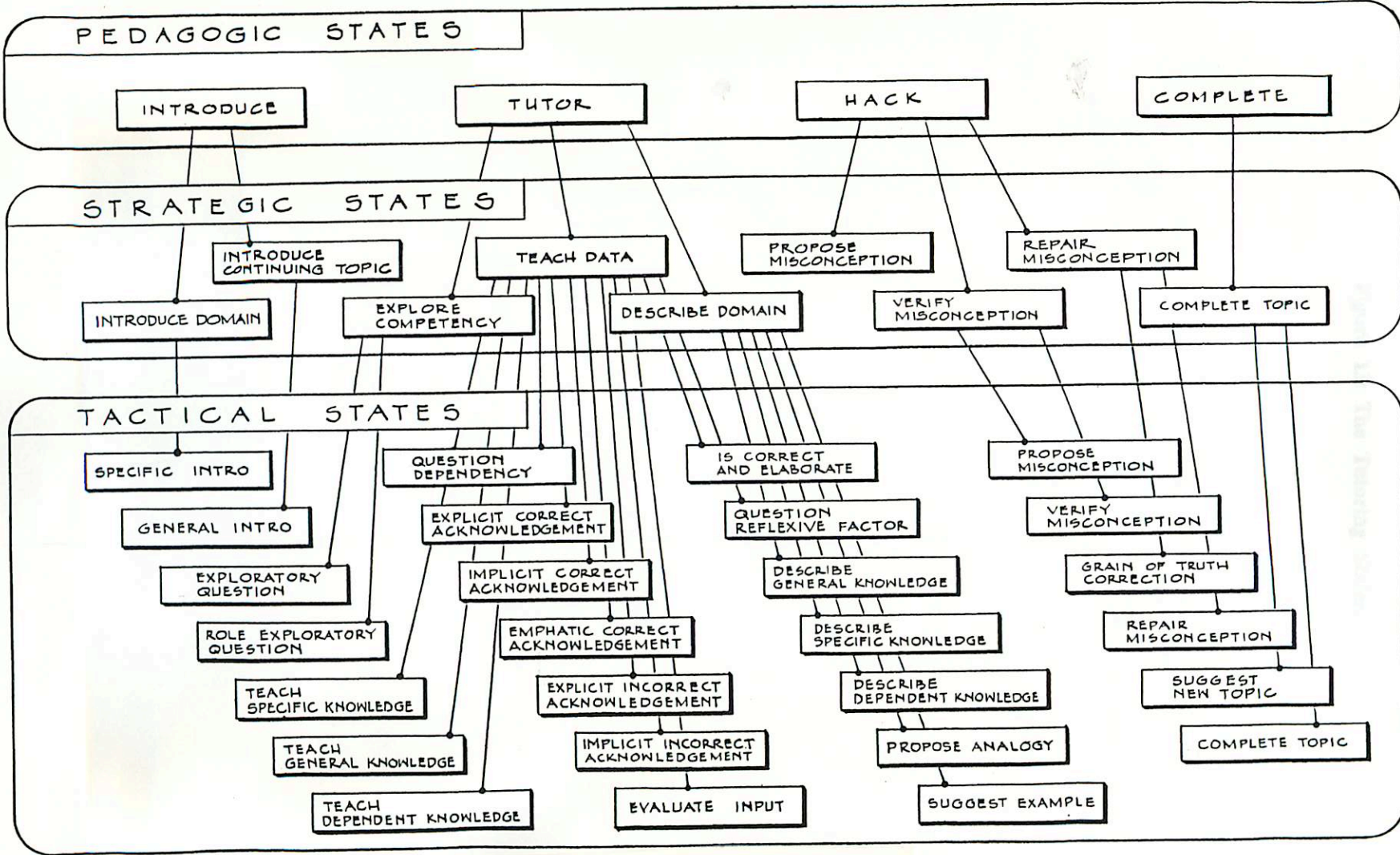


Figure 13: The Tutoring States.

with *elaboration* is entered, causing the utterance “There are rain forests in both Washington and Oregon” to be generated.

---

**\$realize\_specific\_intro**

domain\_model: (complexity connectivity expected\_competency)  
 student\_model: (explicit\_errors implicit\_misconceptions)  
 utter\_elmts: (##introduce\_domain \*major\_topic)  
 next\_states: (\$realize\_evaluate\_input)  
 post\_processing: (record\_student\_response update\_discourse\_model)

**\$realize\_emphatic\_correct\_acknowledgement**

domain\_model: (complexity connectivity expected\_competency)  
 student\_model: (explicit\_errors implicit\_misconceptions)  
 utter\_elmts: (##positive\_strongly ##role\_major\_topic ##brief\_con  
 next\_states: (nil)  
 post\_processing: (complete\_question update\_student\_model)

**\$realize\_grain\_of\_truth\_correction**

domain\_model: (complexity connectivity expected\_competency)  
 student\_model: (explicit\_errors implicit\_misconceptions)  
 utter\_elmts: (##repeat\_user ##negation ##role\_conflict \*misc1  
 next\_states: (\$realize\_suggest\_new\_concept)  
 post\_processing: (complete\_question update\_student\_model)

**Figure 14: Three Typical States at the Tactical Level.**

---

Three typical states at the Tactical Level are given in Figure 14. As shown in the figure, a number of slots is bound to each state and the number and type of slots are common to states at the same level. This consistency facilitates interpretation of states and states at the same level are interpreted in a similar way. The slot Next States defines the next state, the slot Post Processing contains housekeeping functions which are run after the state has been entered, and the slot Utter elmts defines elements used to generate the discourse utterance. The slot Utter Elmts is unique to the Tactical States and contains the essence of the message to be output to the student. The elements from this slots are evaluated and built into a rhetorical specification that is fed to the surface language generator and ultimately output to the student.

The states structure is “nominally” an OR graph with two notable deviations. The first is the set of “default” transitions, indicated by the Next States slots of the state as shown in the example states. These default transitions represent crosslinks through the states that the tutor passes in developing its sequence of actions/characterizations.

The second deviation is generated by meta-rules which functionally capture the shifts observed in classic human tutoring. These rules express the high-level transitions which reflect a switch of the context of the discourse or the system's model of the student's knowledge. The meta-rules preempt the path of the tutor before entrance to a state. At present, the system contains 20 meta-rules. The rules will be described in Chapter 4.

### 3.2.3 The Annotated Knowledge Base.

In order to maintain relevance in the discourse, Meno-tutor uses a knowledge base annotated with pedagogical information (see Figure 15). Annotation enables the system to make decisions about *which* topics to choose and to base that selection on topics linked to previously mentioned topics in the knowledge base. The knowledge base contains coherent as well as inappropriate concepts hierarchically organized and tagged to teaching salience.

This annotation process as viewed by the DMN directs the deep generation of text. In this respect, the methodology of data-base annotations is developed from the work of Conklin [1983], who planned paragraph-length descriptions of visual scenes using an annotated knowledge base where the annotations were derived from an empirically-based consideration of an object's visual *salience*. The annotation process creates a view of the domain "almost as if one could pick up the data base and shake it so that it dangled in order of decreasing salience" [Conklin, 1983, Chapter 3] and serves to highlight elements which, because of size, centrality, or unusualness would naturally be the first topics to be discussed.

Pedagogical annotations in Meno-tutor play a role in tutoring comparable to that of size and centrality in visual pictures: they identify parts of the knowledge base that are most relevant to the student's problem and suggest strategies that will build on what the student already knows. Using the annotated student model is similar to using salience: the model of the student's history is projected onto the general knowledge base and the concepts or examples from which the student would profit are derived based on a match between his history and the contents of the domain. The annotation process provides the planning component with a heuristic for ordering and organizing topics for syntactic realization. The annotations help to individualize the dialogue for each student by selecting the best tutoring strategy for his knowledge level. The hooks into the expert knowledge source, in the form of tutoring information, act as pointers to plausible text generation strategies. In this way pedagogical as well as rhetorical alternatives are tested and identified in an effort to formulate the rules and strategies of tutoring.

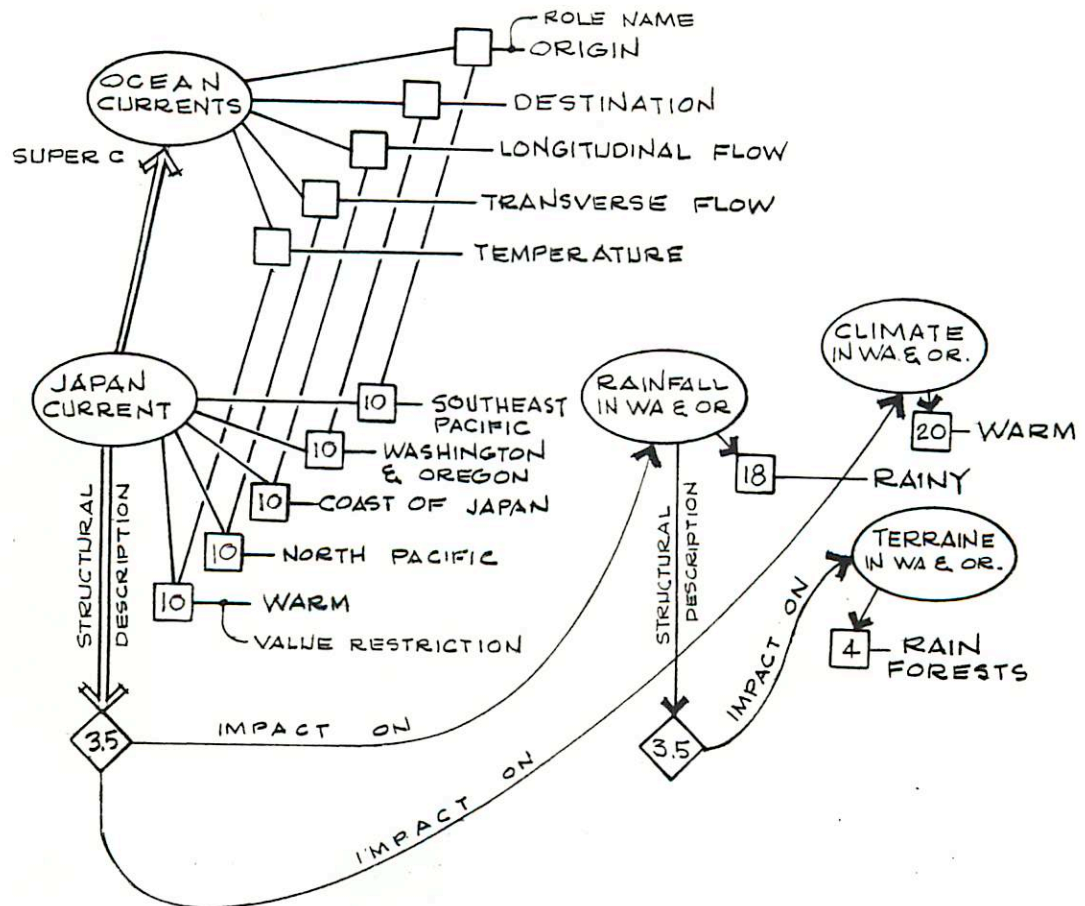


Figure 15: The Expert Knowledge Base for the Rainfall Discourse.

#### The Michelin Rating and Expected Competency.

This pedagogical ordering on the knowledge base is a measure of the depth of a student's knowledge in a domain. It is based on empirical studies of how people learn in the domain and is used to facilitate the choice of topics. The intention has been to organize expert knowledge according to the learner's perception of the information. Two complementary organizing schemes are used to edit and adjust the basic planning mechanism. The annotations define the topics to be discussed, corrected, or questioned and provide planning templates that act as guideposts for searches through the knowledge.



The first annotation scheme, the *Michelin Rating* (MR) [Rissland, 1977], is a number attached to topics in the knowledge base to indicate their intrinsic importance to the tutoring context without regard for the particular student. The numbers range from 0 to 20 and have been chosen by the author, based on empirical studies of learning and teaching in the subject area. The Michelin Rating is equivalent to the importance number used by Collins et al. [1975].

The tutor's knowledge of the student is summarized, per item, as *Expected Competency* (EC), a number which is originally set without knowledge of the user's competency. This number is dynamically updated to indicate the system's view of the student's proficiency. It is primarily a confidence rating signifying the tutor's confidence in its "reading" of the student's knowledge. The value is adjusted upwards as the dialogue progresses; and high values of the EC indicate that the student has some, or at least increasing, knowledge of the domain, while low values indicate that learning has just begun or is not moving ahead rapidly. The EC value may also become lower than the system's default value.

The tutor is on the lookout for deviations from this EC value. If the tutor has made a mistake in the assignment of this value, the student presumably will make errors on topics *below* it. In this case, the tutor might be forced to adjust its EC value downward. On the other hand, if the EC is set appropriately, several correct answers will validate the value of the EC and increase the system's confidence in the value. In this case, a student's single error might be overlooked if the system's confidence in his knowledge is high enough; a student's momentary lapse might therefore, appropriately, be ignored.

The annotated knowledge base guides the tutor's choice of topic; it allows the tutor to avoid inappropriate topics; if topics lie *below* the student's learning threshold they might be too easy and if they lie *above* it they ought to be too hard.

### The Genetic Graph.

The implementation of Meno-tutor and the identification of states at each of three conceptual levels represent a first pass on an implementation of the Genetic Graph (GG) [Goldstein, 1979] (see Section 2.7.1). Goldstein proposed his GG as a separate module to provide the planning and discourse knowledge for a computer coach. His GG contained references between skills the student was trying to learn and contained linguistic cues to enable the coach to generalize from, describe an analogy to, or suggest a deviation from a skill the student already knew.

The advantage of Meno-tutor's annotation system as a means of recording the relevance of, and relation between, topics is that the pedagogical data about domain topics is built right into the representation of domain topics. The annotations can be "tacked on" to an existing expert knowledge base and can be used in collaboration with the planning strategies of a separate teaching module to direct the discourse.

Conversely, the design of the GG requires a separate module, which duplicates the information of the expert knowledge base and combines with it the planning and teaching strategies. In the GG design, each skill must be explicitly represented; in the Meno-tutor design, the expert representation of domain topics is annotated with EC and MR labellings and the teaching strategies, which mediate among the topics, are held separate.

### 3.3 Summary

This chapter described some theoretical and practical issues involved in the generation of discourse and described my experience observing tutoring discourse. These observations have suggested primitives for structuring human-computer discourse (e.g., complex speech patterns and levels of planning and discourse management) and have led to a taxonomy of goals and assumptions made by speakers in the process of discourse.

Some of these discourse primitives and goals have been incorporated into the basic knowledge and planning structures of Meno-tutor. In particular, the tutor used the same three levels of planning strategies observed in human tutoring discourse. The tutor's selection of topics is facilitated by an annotated knowledge base which borrows empirical data about learning and teaching in the target subject area.

## CHAPTER IV

### RULES TO DETERMINE DISCOURSE TRANSITION

Meno-tutor defines a general framework within which tutoring rules can be defined and tested. The mechanisms and data bases discussed in Chapter 3 provide hooks, in the form of tutoring and rhetorical information, from which the system can select plans for text generation. In this chapter, the control structure that arbitrates among the mechanisms to access those hooks, will be described. The basic algorithm and data objects of the system will be described in a technical example along with an explicit view of the way the algorithm, meta-rules and data structures interact during the generation of utterances.

The overall design of the system is briefly reviewed in Section 4.1 and the control structure and meta-rules developed in Section 4.2. A technical example is presented at length in Section 4.3 and Section 4.4 discusses some of issues in the design of the meta-rules. Section 4.4 presents a summary of the chapter.

#### 4.1 The Design of the Tutoring Processes

The design of Meno-tutor establishes two distinct components for the planning and generation of tutoring discourse: 1) the tutoring component, which holds the reasoning and knowledge used to plan the text and 2) the surface language generator, which produces the syntactically correct utterance. The tutoring component makes a decision about what to say and when to say it and passes a specification of the utterance to the surface language generator. The surface language generator takes the specification and produces a syntactically and morphologically correct utterance. These two components interface at the third level of the tutoring component as will be described in Section 4.3.

The decisions of the tutoring component are based on reasoning about several sources of knowledge: expert knowledge, a model of the student's knowledge, and the discourse history. Section 3.2 detailed the three levels of planning that make up the tutoring component; these levels are briefly reviewed here. At the highest level, planning states constrain the discourse to a specific tutoring pedagogy that determines, for instance, how often the system will interrupt the student or how often it will probe him about misconceptions. At this level a choice is made between approaches such as diagnosing student knowledge or introducing a new topic.

At the second level, the planning states further refined the pedagogic plan into a strategy, which specifies a schematic script for the response. For instance, the choice, here, might be between questioning the student or providing a series of examples.



At the lowest level, a tactic is selected to implement the strategy. For instance, if the strategy had been to question the student, the system can choose from half a dozen ways to formulate that question, e.g., it can question the student about a specific topic, or the dependency between topics, or the role of a subtopic. On the other hand, if the strategy had been to respond to the student's answer, the system can choose from among eight ways to respond, e.g., correct the student, or elaborate about the student's answer, or barely acknowledge the answer.

The second process in Meno-tutor is the surface language generator, which receives the specification from the tutoring component and establishes the low level coherency of the utterance. The intended surface language generator is MUMBLE, created by David D. McDonald [1983]. MUMBLE has generated text for other deep level planning systems [Conklin, 1983] and the process is considered fairly straight forward. However, the output from the tutoring component has not yet been sent to MUMBLE. Instead, a simple language generator is used which cuts and pastes text based on definitions provided from a pseudo dictionary, which is built into Meno-tutor. In the example in the section after the next, I describe as many of the steps as necessary to understand the potential interaction between the two systems.

## 4.2 The Control Structure of Meno-tutor

Meno-tutor has a simple algorithm; iterate through states while maintaining data structures and running functions. The algorithm tests meta-rules before evaluating each state; if a meta-rule triggers, the current plan is abandoned and the iteration terminated. If preemption does not occur, each slot in the state is evaluated before the iteration is terminated.

Movement directed by a meta-rule will be called context-dependent; default transitions directed through the states will be called context-independent. Context-dependent transitions, triggered by meta-rules, move Meno-tutor off the default path and onto a path that functionally represents the high-level transitions observed in classic human tutoring as discussed in Section 3.1.2. These transitions are ostensibly more in keeping with student history or discourse history than the default path. The control structure of the tutoring component, passing through the Next State definitions in the slots defines the default transitions. These two forms of control are discussed in this section.

### 4.2.1 Context-Independent Transitions.

Context-independent transitions are predetermined by the arc set of the tutoring states and include the actions designated through the Next States slots. The structure of the tutoring states (Figure 13) is like an augmented transition network (ATN) [Woods,

1970]; it is traversed by an iterative routine that follows a predetermined, though not fixed, path from node to node. As this routine passes through planning levels, beginning at the highest and proceeding to the lowest, it develops a partial plan for the utterance. The plan contains the barest outline of the utterance; it does not, for instance, contain elements of the utterance but rather constrains the number of opportunities available for generation of the utterance at the next level. At the lowest level the specification of the utterance is fairly well defined and encoded in the Utter Elmts slot of the Tactical States.

The importance of the path of the tutor through the tree is not in the nodes themselves, but in the sequence of nodes traversed by it and the ultimate utterance realized at the lowest level. The top-level function, the controller causes routines in the slots of each state to be evaluated. The states and the controller interact with each other in a "Catch 22" cycle: on one hand, the actions of the state perform the necessary housekeeping chores such as recording the "topic described" or the "number of questions asked"; on the other hand, the actions of the states limit the tutor's movement through the state by contributing to the definition of the environment, setting registers, and running functions.

The discourse module, or language generation facility, is activated when the function reaches the third level of the tutoring component. At this level the contents of the slot Utter Elmts are retrieved and sent to the language generation facility.

#### 4.2.2 Context-Dependent Transitions.

The system presently contains 20 meta-rules; most rules originate from more than one state and move the tutor to a single, new state. The preconditions of each rule determine when the tutor will be moved off the default path: preconditions examine data structures such as the student model (e.g., Does the student "know" a given topic?), the discourse model (e.g., Have enough questions been asked on a single topic?), and the domain model (e.g., Do related topics exist?).

The ubiquity of the meta-rules -- the fact that virtually any transition between tutoring states (nodes) can be preempted -- represents a critical deviation from the standard control mechanism of an ATN. The behavior of Meno-tutor might be represented within the definition of an ATN, however including arcs for every meta-rule as part of the arc set of every state would miss the point of the design.

In this section, the meta-rules are first discussed in a representation which is somewhat informal for purposes of explanation. Each rule is then described through explicit code and a brief description detailing the usage of the rule. Finally, in the next section, rules are demonstrated in detail through a technical example.

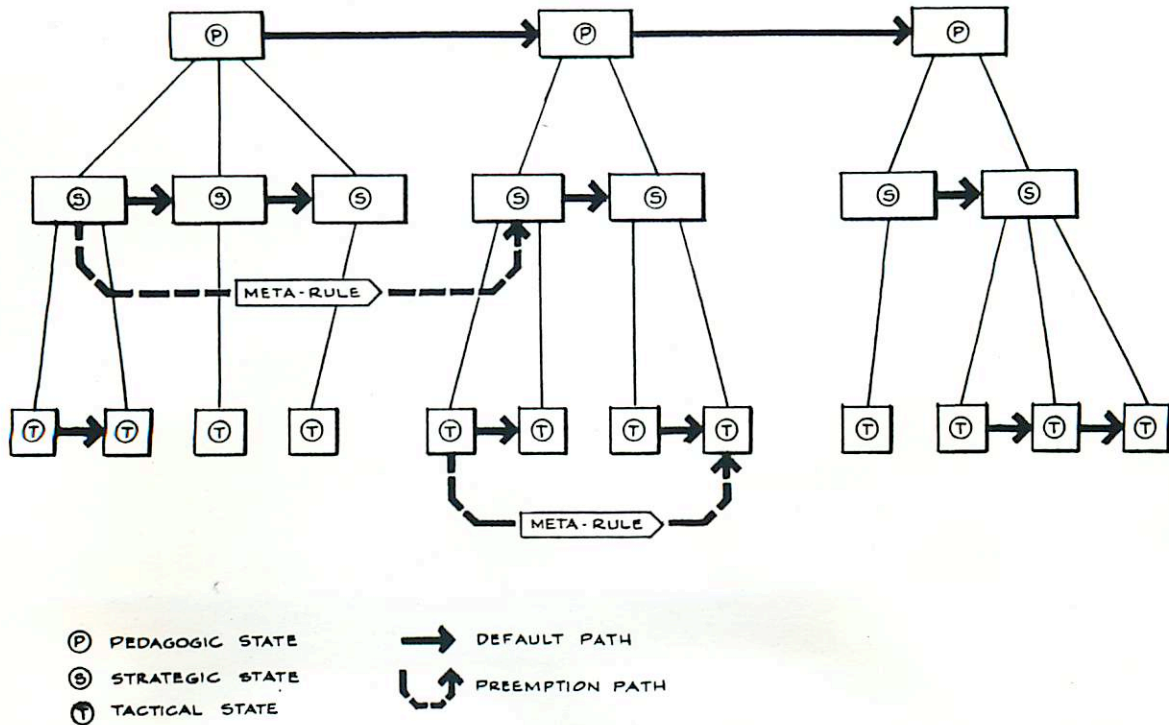


Figure 16: The Action of Meta-Rules.

Figure 17 is an informal representation of meta-rules which describes movement caused by one meta-rule at the Strategic Level, S1-EXPLORE, and another at the Tactical Level, T6-A.IMPLICITLY. In the representation, links joining the prior state to the next state are indicated by the words "From:" and "To:": links join each rule from its prior (original) state to a new one. Entries marked in bold letters indicate specific aspects of the rule. These include:

1. *description* -- a one or two sentence English description detailing the usage of the rule for the use of the designer,
2. *activation* -- a list of registers whose values will cause the rule to fire, and
3. *behavior* -- the observable results of firing the rule.

The activation entry in the informal representation corresponds to two slots in the coded version of the meta-rules, precondition and prior state; the other two entries describe the use and behavior of the rule as encoded in the other two slots. The slots of the coded

**S1-EXPLORE – a Strategic Meta-rule**

From: teach-data  
 To: explore-competency

**Description:** Moves the tutor to question the student about about a variety of topics.

**Activation:** The present topic is complete, but the tutor has little confidence in its assessment of the student's knowledge.

**Behavior:** Generates an expository shift from detailed examination of a single topic, to a shallow examination of a number of topics.

---

**T6-A.IMPLICITLY – a Tactical Meta-rule**

From: explicit-incorrect-acknowledgement  
 To: implicit-incorrect-acknowledgement

**Description:** Moves the tutor to utter a brief acknowledgement of an incorrect answer.

**Activation:** The wrong answer threshold has been reached and the student seems confused.

**Behavior:** Shifts the discourse from a explicit correction and explanation of an incorrect answer to a simple response that recognizes, but does not dwell on, the incorrect answer.

---

**Figure 17: Informal Notation of a Meta-Rule.**

---

version of the meta-rule will be discussed now.

Each meta-rule is implemented as a LISP structure with four slots which hold various parts of the rule. The slots are interpreted in a fixed sequence. Application of a meta-rule (see Figure 18) includes a test for satisfaction of Prior State and Precondition, an Action to be performed if the first two predicates are satisfied, and a list of Postprocessing functions, which are run after the rule is triggered. At the present time, the Post Processing slots are not filled and most housekeeping chores are handled by the Post Processing slot of the state to which the tutor has been moved. However, the meta-rule slot is available for the future placement of functions which might, for instance, record the number of high-level discourse shifts or interpret the effectiveness of the discourse as a function of the number of discourse shifts.



The Preconditions contain predicates which examine elements of the student model (e.g., Does the student "know" a given topic?), the discourse model (e.g., Have enough questions been asked on a single topic?), and the domain model (e.g., Do related topics exist?). All Preconditions must be satisfied for a rule to be run. Once the tutor is in a Prior State from which the meta-rule can fire, and the Preconditions are satisfied, the rule will be *enabled*. If the rule is run the state of the tutor will be changed, the Postprocess routines would update the student or discourse model and the current iteration of the system terminated.

---

1. *Check Prior state:* Check that the tutor is in a state defined by the list of Prior States.
2. *Check the Precondition of the meta-rule:* Check that all the predicates of the rule are satisfied. The predicates test attributes of the student or discourse model (e.g., length of interaction, evidence of misconception, or measure of domain complexity).
3. *Apply the meta-rule:* Move the tutor into the new state indicated by the rule or shift the range of topics.
4. *Evaluate the Post Processing functions:* Run functions stored in the meta-rule to update the discourse or student model.

**Figure 18: Application of a rule.**

---

Several rules may be enabled at once; only one will be triggered and run. The selection of a rule to run is made by taking the first element of a list of enabled rules. In this implementation, the order of evaluation of the rules, which is also the order of the enablement of a rule, is roughly based on the inverse popularity: i.e., the least popular rule is tested first. If the most popular rules were tested first, they would be triggered frequently, effectively blocking out the least popular rule.

In the next six figures the meta-rules of Meno-tutor are presented first in code, pretty printed as LISP structures and then in a brief description detailing the Prior State and Preconditions which must be satisfied for their enablement. In the code, the state entered by the tutor after the rule is enabled is listed in the slot Action and the states from which the rule can be enabled are listed in the slot Prior State. The slot Post Processing is not listed in the figure, but is part of the actual LISP code.

The rules set forth here reflect only one of several possible interpretations of the knowledge underlying tutoring discourse. These rules are not presented as a definite set, but rather as a useful set given my current level of understanding of tutorial discourse.

**P2-HACK**

```
preconditions: (*misconception_triggered *topics_complete *completion_acknow
prior_state: ($complete $tutor)
action: (setq next_states ($hack_misconception))
```

**P3-COMPLETE**

```
preconditions: (*topics_complete *competence_confirmed)
prior_state: ($tutor $hack_misconception)
action: (setq next_states ($complete))
```

**P4-TUTOR**

```
preconditions: nil
prior_state: ($hack_misconception)
action: (setq next_states `($tutor))
```

**Figure 19: Pretty Printing of the Code for Pedagogic Meta-rules.**

**P2-HACK:** Produces a high level structural shift to the state *hack misconception* from the states *tutor* or *complete* a topic. It causes the tutor to produce a series of questions which suggest, verify or repair a possible misconceptions. The Preconditions for this rule is a test for three conditions: *\*misconception\_triggered*, *\*topics\_complete* and *\*completion\_acknowledged*. The first is set when a student's response is linked with an misconception in the expert knowledge base, i.e., an explicit error is linked to a set of possible misconceptions at the time the expert knowledge base is built. *\*Topics\_complete* is toggled when new topics presented by the tutor have been either answered correctly by the student or corrected by the tutor. *\*Completion\_acknowledged* is toggled when the tutor indicates that the current topic is finished.

**P3-COMPLETE:** Produces a shift in pedagogy away from tutoring or hacking at a misconception and towards an approach which *completes* the current topic. The movement causes the tutor to sum up one topic or to designate a new set of topics for discussion. This rule is triggered after two registers have been triggered: *\*topics\_complete* and *\*competence\_confirmed*. The first is triggered when the all topics introduced by the tutor have been either answered satisfactorily or corrected by the tutor. The second register is triggered as a function of *#topics\_list* and *#topics\_questioned*, both of which are entries in the property list of the variable *\*counters*.

**P4-TUTOR:** Produces a shift from *hacking* at a misconception, towards *tutoring*, i.e, the tutor will explore the student's competency, teach data from the expert knowledge base or describe the domain, perhaps through examples and analogies. This rule has not preconditions and is the last pedagogical meta-rule to be tested.

**Figure 20: A Description of the Pedagogic Meta-Rules.**

```

S1-EXPLORE
  preconditions: (*question_complete *related_topics)
  prior_state:  ($teach_data)
  action:       (setq next_states `($explore_competency))

S2-TEACH
  preconditions: (*topics_complete *related_topics)
  prior_state:  ($introduce_domain $explore_competency $describe_domain)
  action:       (setq next_states `($teach_knowledge))

S3-DESCRIBE
  preconditions: (*wrong_answer_threshold *salient_topic)
  prior_state:  ($explore_competency $teach_knowledge)
  action:       (setq next_states `($describe_domain))

S4-VERIFY
  preconditions: (*suspicion_confirmed)
  prior_state:  ($propose_misconception)
  action:       (setq next_states `($verify_misconception))

S5-REPAIR
  preconditions: (*misconception_verified)
  prior_state:  ($verify_misconception)
  action:       (setq next_states `($repair_misconception))

S6-JETTISON
  preconditions: (*terminate_strategy)
  prior_state:  ($teach_data $complete_topic)
  action:       (setq next_states
                `(pop_planning_level qstate `$$strat_state))

```

**Figure 21: Pretty Printing of the Code of the Strategic Meta-Rules.**



- S1-EXPLORE:** Creates a rhetorical shift from a detailed discussion of a topic to a series of questions which deal, in a shallow way, with a variety of topics. Is triggered when the register *\*question\_complete* is true and *\*related\_topics* remain to be *explored*. This rule moves the tutor from a strategy of *teaching data* to one in which the student's threshold of knowledge is explored.
- S2-TEACH:** Possibly the most universal strategic rule; it is triggered from any of the other strategic states under the Pedagogic state to *tutor*. This rule requires the completion of all prior topics, *\*topics\_complete*, and the existence of a *\*related\_topic*, which will be discussed. This rule causes the tutor to begin to discuss or question the related roles of a topic.
- S3-DESCRIBE:** This rule is selected sparingly and requires evidence that the student is confused, the *\*wrong\_answer\_threshold* has been reached and that a *\*salient\_topic* exists, a topic with a high Michelin Rating (see Section 3.2). The rule causes the tutor to discontinue a question-answer routine and to begin a lengthy description of data which might be important to the current topic. The salient data is described, rather than questioned, because the previous strategy has proved ineffective, as evidenced by the fact that the *\*wrong\_answer\_threshold* has been met. The rule is triggered from either *explore strategy* or *teach domain*.
- S4-VERIFY:** This rule is selected only after the tutor has questioned the student about an underlying definition or belief. If the student's answer confirms the possibility that he has a misconception, that is, if *\*suspicion\_confirmed* is set, the tutor will move to a state where it can generate another question to confirm or refute the existence of the misconception.
- S5-REPAIR:** This rule is selected only after the misconception has been proposed and verified. If the student's answers confirm that he has the misconception, the tutor moves to a state where it can repair, i.e., help the student to restructure, his knowledge eliminating the misconception.
- S6-JETTISON:** This rule is triggered if the register *\*terminate\_strategy* is set as a part of the discourse history model. *\*Terminate\_strategy* is set under two conditions: 1) all the topics have been discussed, or *\*topics\_complete* is true, and the model of student knowledge is well established, or *\*competence\_confirmed* is true, and 2) the tutor has stated that the present discussion ought to be terminated and a new one begun. The meta-rule removes the tutor from the Strategic Level of planning and moves it to the Pedagogic Level where a new round of decisions making begins. This rule, then, is triggered at a turning point in the conversation, such as, after the model of the student is well established or a topic terminated.

Figure 22: A Description of the Strategic Meta-Rules.

**T1-A.CORRECT**

preconditions: (response\_is\_correct)  
 prior\_state: (\$realize\_evaluate\_input)  
 action: (setq next\_states  
 `(\$realize\_explicit\_correct\_acknowledgement))

**T2-A.INCORRECT**

preconditions: (response\_is\_incorrect)  
 prior\_state: (\$realize\_evaluate\_input)  
 action: (setq next\_states  
 `(\$realize\_explicit\_incorrect\_acknowledgement))

**T3-ELABORATE**

preconditions: (\*question\_complete \*additional\_roles\_exist response\_is\_correct)  
 prior\_state: (\$realize\_explicit\_correct\_acknowledgement)  
 action: (setq next\_states  
 `(\$realize\_is\_correct\_and\_elaborate))

**T4-Q.DEPENDENCY**

preconditions: (\*dependency\_exists)  
 prior\_state: (\$realize\_question)  
 action: (setq next\_states  
 `(\$realize\_question\_dependency))

**T5-JETTISON**

preconditions: (response\_is\_incorrect \*misconception\_triggered)  
 prior\_state: (\$realize\_evaluate\_input)  
 action: (setq next\_states  
 `(pop\_planning\_level qstate `\$\$stact\_state))

**T6-A.IMPLICITLY**

preconditions: (\*wrong\_answer\_threshold \*student\_in\_doubt)  
 prior\_state: (\$realize\_explicit\_incorrect\_acknowledgement)  
 action: (setq next\_states  
 `(\$realize\_implicit\_incorrect\_acknowledgement))

**T7-D.IMPACTOR**

preconditions: (\*wrong\_answer\_threshold)  
 prior\_state: (\$realize\_describe\_general\_knowledge)  
 action: (setq next\_states  
 `(\$realize\_dependent\_knowledge))

**T8-T.SPECIFIC**

preconditions: (\*question\_complete \*answer\_available)  
 prior\_state: (\$realize\_explicit\_incorrect\_acknowledgement)  
 action: (setq next\_states  
 `(\$realize\_teach\_specific\_knowledge))

**T9-A.EMPHATIC**

```
preconditions: (*correct_after_wrong_answer)
prior_state:  ($realize_evaluate_input)
action:      (setq next_states
              `($realize_emphatic_correct_acknowledgement))
```

**T10-R.EXPLORE**

```
preconditions: ($functional_reasoning response_is_correct)
prior_state:  ($realize_exploratory_question)
action:      (setq next_states
              `($realize_role_exploratory_question))
```

**Figure 23: Pretty Printing of the Code for the Tactical Meta-Rules.**

- T1-A.CORRECT:** Generates an explicit acknowledgement of a correct student answer. It is triggered from the state \$realize\_evaluate\_input, which receives and evaluates the student's response and is enabled only when the register response\_is\_correct is toggled. It may be argued that this rule and the next, T2-A.INCORRECT, are not true meta-rules—their actions could be effected, for instance by a simple production rule. Though this is true, it misses the point of the meta-rule definition. Having these two rules allows the system, and the author, to recognize and record events that occur around the discourse transition to repond to a correct answer.
- T2-A.INCORRECT:** Is triggered by the student's incorrect answer and moves the tutor to explicitly acknowledge the correct answer (see T1-A.CORRECT, above).
- T3-ELABORATE:** Elaborates upon the standard acknowledgement to the student's correct answer and provides additional information about a role or attribute of the topic. This rule requires that previous topics have been correctly answered or completed, i.e., the register \*question\_complete is set, that related topics remain to be described, \*additional\_roles\_exist is set, and that the student's answer was correct, response\_is\_correct is true.
- T4-Q.DEPENDENT:** Is enabled from the Tactical State \$realize\_question after a topic has been described and questioned. It is triggered only if a dependency exists between the current topic and a prior one as reflected set register \*dependency\_exists. Produces a question about the effect of the current topic on the prior one.
- T5-JETTISON:** Triggered by an incorrect student answer and by evidence of a misconception, i.e., \*misconception\_triggered is set. The rule can be enabled only if the student's wrong answer has been linked to a stored misconception. If run, the meta-rule will remove the tutor from planning at the Tactical Level and move it to additional planning at the Strategic Level, to consider the underlying misconception.
- T6-A.IMPLICITLY:** Generates an implicit, noncommittal response to the student's wrong answer. The rule is triggered after the current strategy has failed, as evidenced by the fact that the number of permitted wrong answers is met, i.e., the register \*wrong\_answer\_threshold is set, or the student seems confused, \*student\_in\_doubt is set. The rule is triggered from the state *evaluate input*, after the tutor has received and evaluated the student's answer.
- T7-D.IMPACTOR:** This rule generates a description of a topic which is a factor in the existence or function of the current topic (e.g., the Japan Current is a causal factor on the production of rainfall in Washington and Oregon). The triggered rule produces additional material upon which the student might make an inference about the current topic. The rule is triggered if the student is in doubt, i.e., the register \*wrong\_answer\_threshold is set, and can only be triggered from the state *describe general knowledge*.

**T8-T.SPECIFIC:** This rule is triggered after the tutor has acknowledged an incorrect answer and there is specific data about the current topic to be used in the explanation, i.e., the register *\*answer\_available* is set. It is triggered after previous topics have been adequately described or corrected, i.e., the register *\*question\_complete* is true. The rule causes generation of a statement about a specific fact in the expert data base.

**T9-A.EMPHATIC:** This rule refines an explicit acknowledgement of the correct answer. It is triggered after the student has answered a question correctly, but has answered a previous set of questions incorrectly, implying that he may have doubts about the subject area. The rule causes the generation of additional material about the role or attribute of the latest topic.

**T10-R.EXPLORE:** Generates a description of the function or use of a topic after questioning the student about the topic itself. The rule is triggered if the student's answer about the topic is correct, yet understanding the topic involves understanding its function as well as its definition, i.e., the register *\$functional\_reasoning* is set.

**Figure 24: A Description of the Tactical Meta-Rules.**

The adequacy of these rules can be fully tested only by extensive use of the system to produce many discourses. Empirical studies of human-machine discourse using these rules will almost certainly supplement as well as correct the details of the "untested" rules. As others work with these rules, I expect the quality and refinement of the rules to evolve.

### 4.3 An Example Discourse

In this section I provide a more explicit view of the way the algorithm, meta-rules and data structures interact during the generation of utterances. I describe the operations of the tutor and provide snapshots of its passage through the states while listing the text uttered by the system. At the same time I list as much of the current context as is relevant for each step of text generation.

---

**TUTOR:** No, it is rather mild. Can you guess about the rainfall there?

**STUDENT:** Normal, I guess.

**TUTOR:** Well, the Japan Current, which starts in the south-eastern Pacific, flows along the coast of Japan and across the North Pacific, ending up off the coast of Washington and Oregon. How do you think that current affects the climate there?

**Figure 25: The Discourse Snippet.**

---

The discourse used for this example is a snippet of the rainfall discourse in Figure 1 on page 6 reproduced here in Figure 25. The portion of the tutoring space covered by the example discourse is given in Figure 26 and will be explained in detail in the following discussion. The expert knowledge base is the KL-ONE network shown in Figure 15 (page 51).

The discourse begins after the student has failed to answer the first question correctly. Since the system does not as yet perform natural language understanding, we simulate the student's answer and the system matches the input against the correct answer, which is stored in the knowledge base. The system prepares to respond by explicitly acknowledging the student's error.

#### 4.3.1 Explicit Incorrect Acknowledge Tactic.

The code in Figure 28 shows the top-level algorithm and the snapshot in Figure 27 provides a snapshot of the tutor's passage through a small portion of the tutoring component. In the algorithm, the function *Memo-tutor* initializes the discourse and calls the routine *the-controller* which iterates on the states. In the snapshot, the tutor begins in the state *explicit incorrect acknowledgement*. Being a tactical state, the principal action this state takes is to utter the default response.

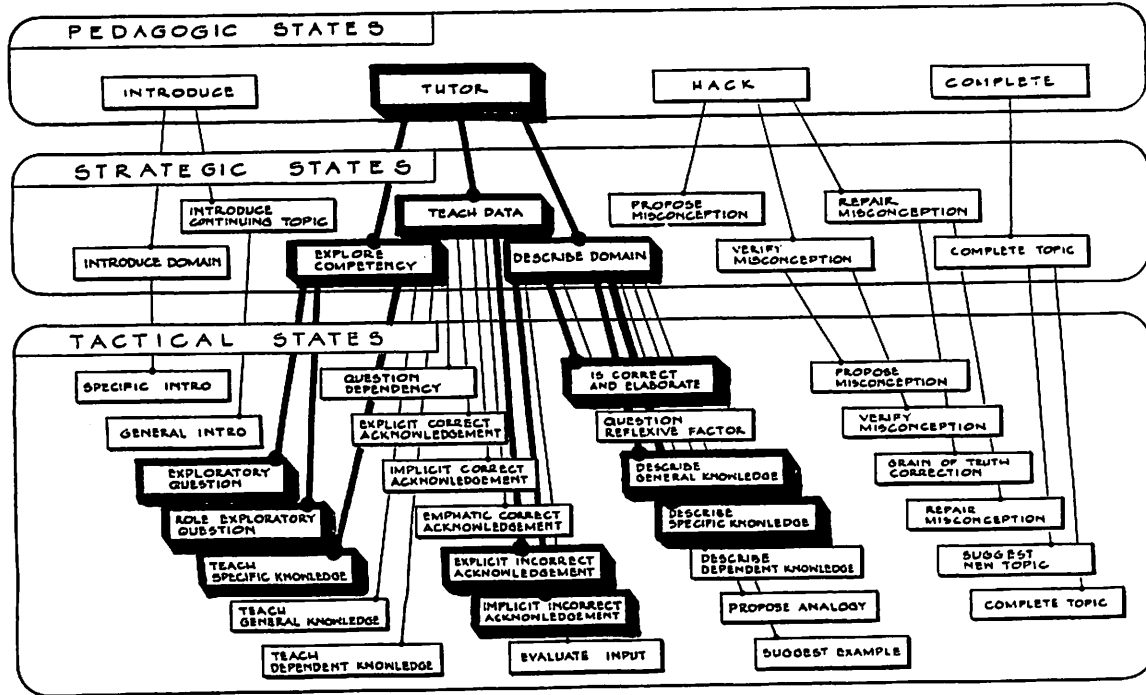


Figure 26: The Portion of the Tutoring Structure used in the Example.

In the first iteration of *the-controller*, the function process-node tests the meta-rules to verify that the default utterance should be produced. The function process-node (Figure 28), in evaluating meta-rules and states, is the first routine to respond to the context of the discourse, i.e., the first to sample student and discourse history. It calls *preempt-path*, in the first line of the *cond* command; if a rule is triggered, it abandons the current state and binds a new state for as current state. As will be explained below, no meta-rule fires during this first iteration and process-node moves ahead to evaluate the default state *explicit incorrect acknowledge*.

Since the default state, *explicit incorrect acknowledgement*, is a Tactical state, the third entry in the case inside the *cond* statement is run. This evaluation involves construction of a specification for machine response. Control then enters the function *realize utterance* which retrieves the contents of the slot descriptor *Utter Elmts*. These contents are packaged into a rhetorical specification and would be sent to the surface language generator and rendered into a legitimate utterance.



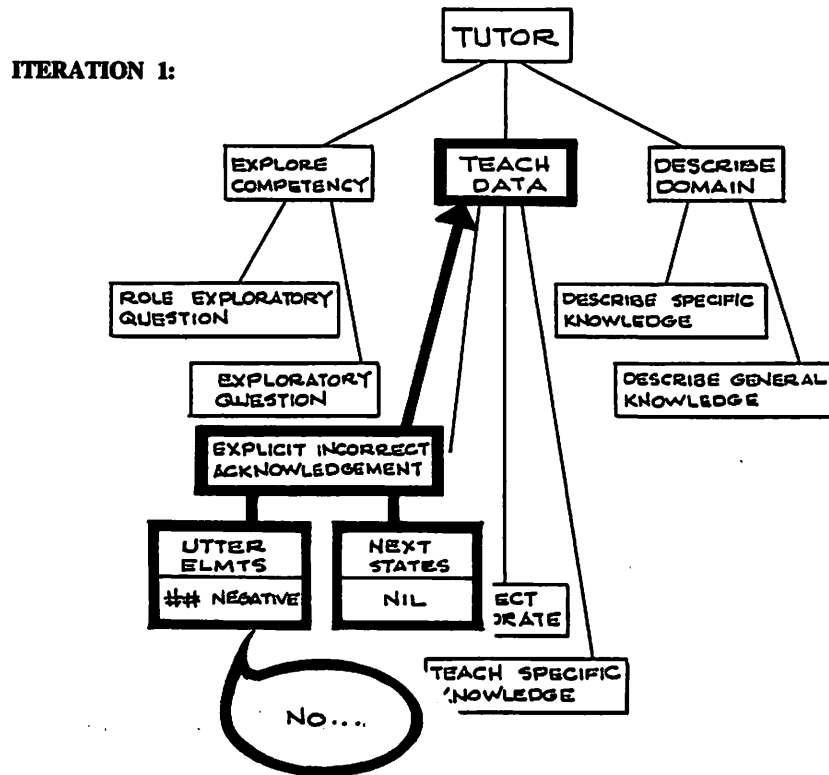


Figure 27: Explicit Acknowledgement Snapshot.

```

Meno_tutor
  (initialize_discourse)
  (the_controller current_state)

the_controller (state)
  (repeat (*ply_count)
    (process-node state)
    (setq state (car next_state)
      *ply_count (add1 *ply_count)
    )
  )
  until (eq *ply_count *end_dialogue))
  (terpri)
  (**END**CONTROLLER**)
  
```

Figure 28: Code for Meno-tutor and The Controller.

```

process_node (state)
  (let (level (get_structure_type state))
    (cond ((preempt_path state level)
      (t (case level
        (('$$ped_state) (execute_state state))
        (('$$strat_state) (execute_state state))
        (('$$tact_state)
          (realize_utterance state)
          (evalaute_slot_post_process
            (get_slot_access state 'post_processing)
            state)
          (evalaute_slot_next_state (get_next_state state))))
        (format " inappropriate state " state)))) ; cond
    (if (null next_state)(pop_planning_level state level))) ; let

```

Figure 29: Code for Process Node.

As mentioned above (Section 4.1) the actions of the surface language generator are simulated. If MUMBLE, were used, each element to be used in the specification would have been defined as a rhetorical instruction in MUMBLE's dictionary developed specifically for tutoring about rainfall topics. Each instruction would define a goal that is interpreted as a program of decisions to be made and is enumerated as a hierarchy of relations in a fixed vocabulary. The text produced would be realized by satisfying each goal according to the decision-procedure given by the dictionary entry.

In the function `realize_utterance`, Meno-tutor handles the work normally handled by MUMBLE. The element `**negation` is retrieved from the slot `Utter Elmts`, where it's rhetorical-instruction in the pseudo-dictionary is processed and a phrase such as "That's wrong . . ." or "No", or "That's not true." is produced. In this example, the utterance produced by Meno-tutor is

"No. . . ."

Having said this, the tutor still has "control" of the discourse and can continue to elaborate its response to the student's wrong answer. After the text is uttered, routines in the Post Processing slot handle the housekeeping chores. The first register to be updated is `#wrong_answers`, located on the property list of the register `*counters`, which holds a variety of registers such as `#right_answers`, `#wrong_answers`, `#topics_descibed`, and `#topics_exampld`. Next, the register `Next State` is assigned the value of the contents of the slot `Next States` and the case statement is exited.

In the present design there is no default path out of *explicit-incorrect-acknowledgement* at the tactical level. That is, the value of the `Next State` slot, which defines the default movement, is nil. With a different set of rules and a different value in this slot, the tutor might, for example, have continued speaking or might have made

```

pop_planning_level (state level)
  (setq next_state
    (list (search_parental_units state
      (find_state_descriptor state level))))

execute_state (state)
  (setq next_states (plist state))

```

**Figure 30: The Code for Pop Planning Level and Evaluate State.**

a comment about the student's answer, or perhaps elaborated on a portion of it. However I decided, in designing the rules, that the better things to do at this point is to move to a higher planning level and to consider reformulating either the strategy or the pedagogy of the utterance, as indicated in the figure by the up arrow from this state to the parent state, *teach data*. (The parent state is that unique state directly above and connected to the current state.)

After *pop-planning-level* terminates, control returns to the function *process-node*, which also terminates. At this point, the controller continues execution and tests whether the discourse should be terminated. The variable *\*end\_dialogue* was set to 20, i.e., the tutor will produce 20 utterances, and since this number is not yet reached, the discourse continues. The entire high-level algorithm, presented in Figure 31, is begun again with *teach data*, defined in the prior iteration, as the new current state.

#### 4.3.2 Teach Data Strategy.

Once in *teach-data* we take the default path back down to the tactical level to *teach-specific-data*. Again, at this point a meta-rule might have applied to take the tutor to a more particular tactical state, but in this case that did not happen. The algorithm for the next few iterations is highlighted along with a snapshot which sketches passage of the tutor through a small portion of the Discourse Management Network ("DMN") (see Figure 32).

The function *execute-state* in the case statement of *process-node* returns *teach specific data* as the next tactical state. *Execute state* is a straightforward function that returns the first in a series of states to implement the higher-level state, which was its argument. *Execute state* acts as a hook for more sophisticated routines which might ultimately be used to select the optimal lower-level state based on student or discourse history. For instance, the selection might be predicated on a differential model using expert rules to select strategies or tactics based on type of domain. Or it might be biased towards selection of methods that have worked well with a particular student.

Meno-tutor

```
(initialize_discourse)
(the_controller current_state)
```

the\_controller (state)

```
(repeat (*ply_count)
  (process_node state)
  (setq state (car next_state)
            *ply_count (add1 *ply_count))
  until (eq *ply_count *end_dialogue))
(terpri)
ENDCONTROLLER
```

process\_node (state)

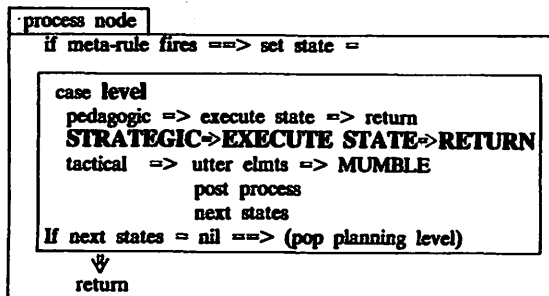
```
(let (level (get_structure_type state))
  (cond ((preempt_path state level)
        (t (case level
             (('ped_state) (execute_state state))
             (('strat_state) (execute_state state))
             (('stact_state)
              (realize_utterance state)
              (evalaute_slot_post_process
               (get_slot_access state "post_processing"
                state)
              (evalaute_slot_next_state (get_next_state state))))
             (format " inappropriate state " state)))) ; cond
  (if (null next_state)(pop_planning_level state level))) ; let
```

preempt\_path (state level)

```
(if (memq state list_preemptable_state)
  (let* (enabled_rules (check_preconditions (check_prior_state state level))
        triggered_one (condcar enabled_rules))
    (if triggered_one (act_on_transition triggered_one) (reset_discourse_model triggered_one)
    triggered_one))
```

**Figure 31: The High-level Algorithm for Generating Tutoring Discourse.**

## ITERATION 2:



## ITERATION 3:

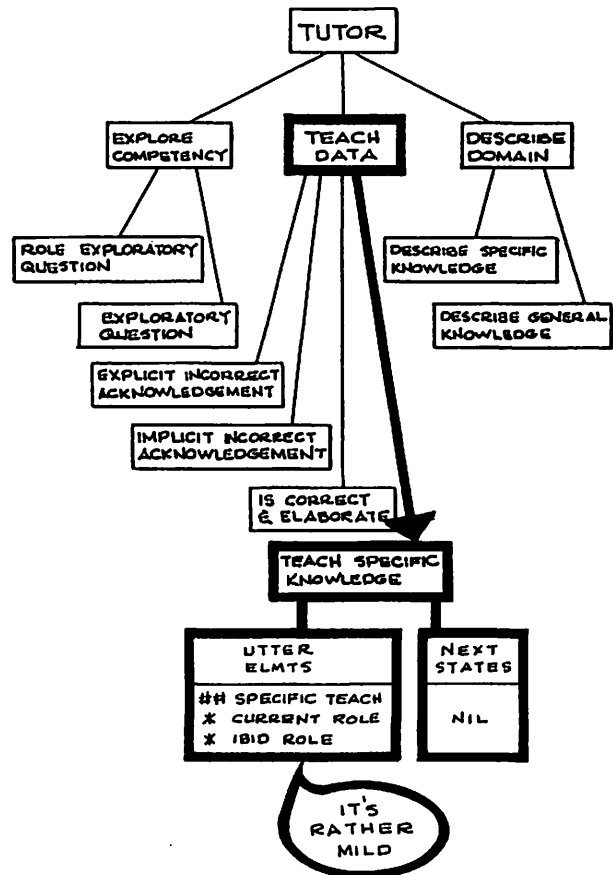
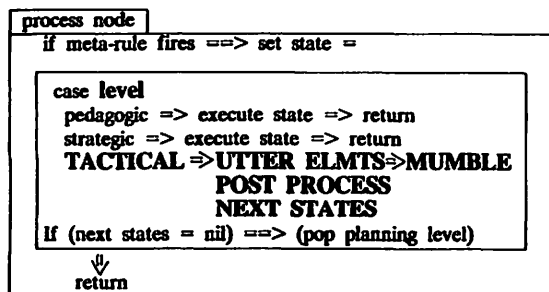


Figure 32: Teach to Teach Specific Snapshot.

At *teach specific data*, as in any tactical state, the tutor says something, in this case extending the utterance already begun with “No”. The utterance is constructed from the specification that always goes with this decision unit, but as individualized by the values its elements have at this point in the discourse and in this domain. The first element of the utterance is designated as *specific-value (current-topic)*, where *current-topic* has been carried forward from the previous ply of the discourse and is still “climate in Washington and Oregon”. Its specific value is “rather mild” (a canned phrase), and the surface language generator (MUMBLE [1983]) renders it in this discourse context (i.e., “full sentence”) as

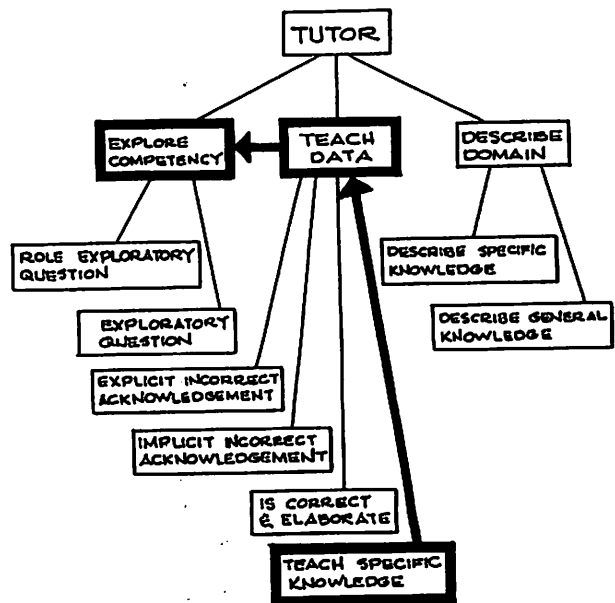
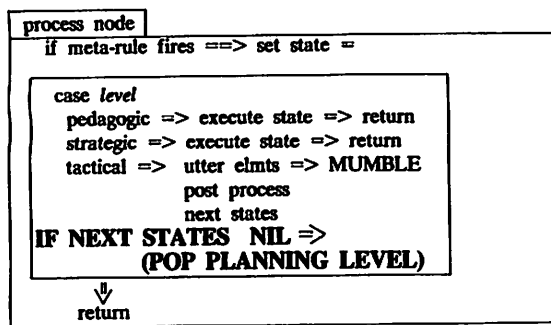
“No, it’s rather mild.”

### 4.3.3 The Explore Strategy.

From *teach specific knowledge* there is no default path and so the tutor again moves up to *teach data* (Figure 33). This time, however, the context has changed and a meta-rule takes the tutor to a different decision-unit than the last time.

The strategies available at this time are *explore competency*, *describe domain*, and *teach data*. Each strategy expresses the pedagogy in a different way and each places emphasis on a different goal. For instance, *describe domain* provides lengthy description, analogies or examples about the current topic, *teach data* asks brief questions or explains the topic, and *explore competency* surveys the student's knowledge about a variety of topics.

ITERATION 4:



ITERATION 5:

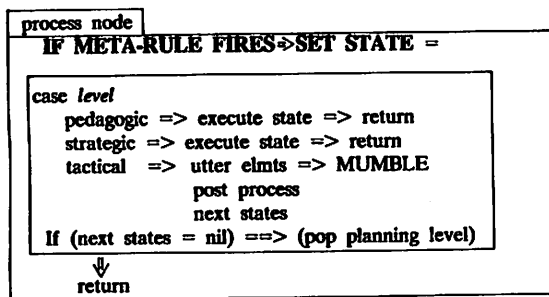


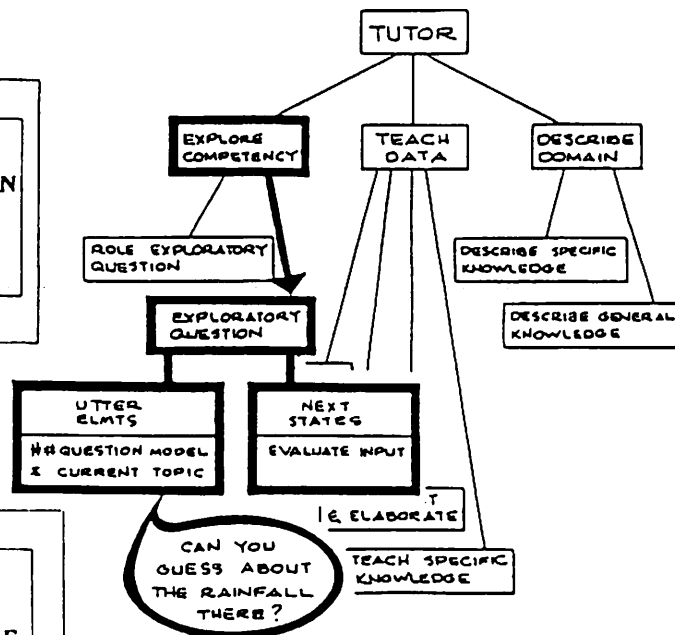
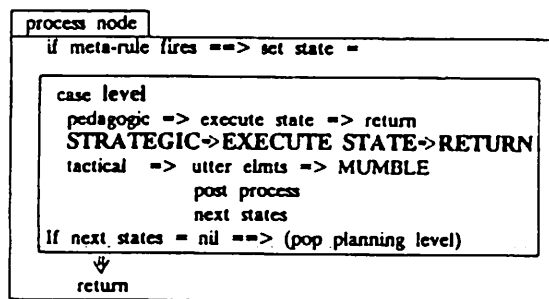
Figure 33: Teach to Explore Snapshot.

The context has changed because all the topics brought up till this point in the discourse have been answered or resolved. In detail, what happened when the tutor supplied the correct answer to its own question (i.e., "Its rather mild"), the DMN register \*question\_complete was set, satisfying one of the preconditions of the meta-rule S1-EXPLORE (Figure 17). Its other precondition was already satisfied, namely that another topic related to current topic remains to be discussed (indicated by a register). At this point in the discourse, with only one question asked, the model of student knowledge is judged to be incomplete. This judgement is made as a function of two

registers, #topics\_list and #topics\_questioned, both properties on the property list of the register \*counters.

When S1-EXPLORE triggers it moves the tutor to the state *explore competency*, in effect establishing that the previous topics are complete and that a new topic can be explored. The next most salient topic in the knowledge base is "rainfall in Washington and Oregon" and it becomes the current topic.

ITERATION 6:



ITERATION 7:

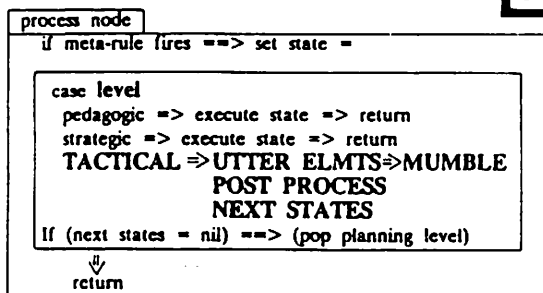


Figure 34: Explore to Exploratory Question Snapshot.

Once in *explore-competency*, the tutor takes a default path to the tactical level and to *exploratory-question* (Figure 34) where it asks another question on a new topic at the threshold of the student's knowledge. The utterance this time is constructed from the specification question-model (current-topic). Question-model would be an entry in the dictionary of the surface language generator that realizes a question about the student's "picture" or "view" of the data: "What do you think about . . .", "Can you guess about . . .", and "Do you believe that . . ." Current-topic as mentioned above was rebound to "rainfall in Washinton and Oregon" in the previous iteration. The rhetorical specification generated by Meno-tutor is:

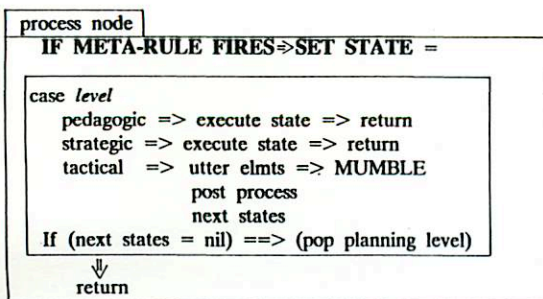
"Can you guess about the rainfall there?"

At this point Meno-tutor continues along a default path and enters the tactical state *evaluate-input* (not shown) which receives and evaluates the student's answer. This answer is again wrong and the default path moves the tutor, once again to *explicit-incorrect-acknowledgement*, where it would normally correct the student, as done once before.

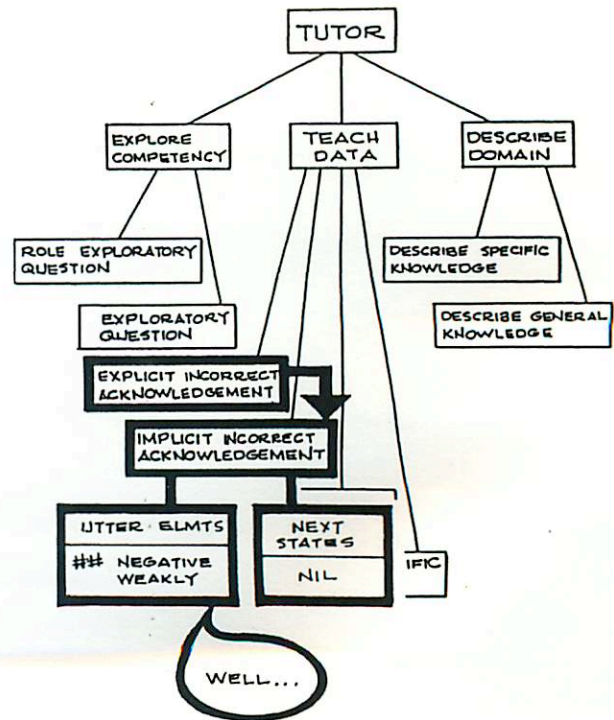
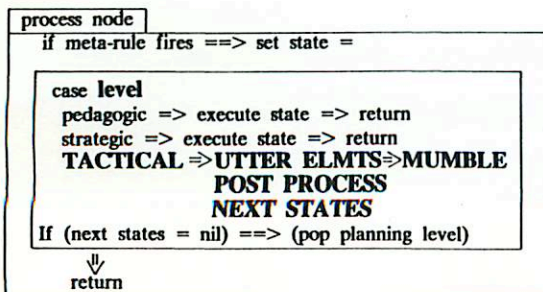
**4.3.4 The Implicit Acknowledge Tactic.**

However, this state is not evaluated because context is now different and a new meta-rule, T6-A.IMPLICITLY (Figure 17) fires moving the tutor to another decision-unit (Figure 35).

ITERATION 8:



ITERATION 9:



**Figure 35: Explicit to Implicit Incorrect Acknowledgement.**

The difference in context is two-fold: 1) the student seems confused and 2) the test for wrong answers threshold is met. Recognizing a confused student is admittedly a subjective and imprecise inference for a machine tutor. In this implementation, we have

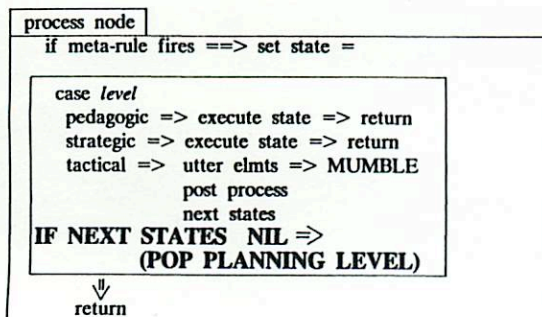


chosen to measure it as a function of the number of questions asked, the number of incorrect responses given, and the extent to which the student's frontier of knowledge has been explored. In the example discourse, two questions have been asked, two answers have been answered incorrectly, and the student's frontier of knowledge is barely explored. Therefore, the student is judged to be confused and the meta-rule T6-A.IMPLICITLY is triggered, forcing the system to move to the tactical state *implicit-incorrect-acknowledgement*. Instead of correcting the student, this state causes a response which implicitly recognizes, but does not dwell on, the incorrect answer. The tutor responds with

"Well, . . ."

There is no default path from *implicit-incorrect-acknowledgement* and the tutor once again moves up to *teach-data*, Figure 36.

#### ITERATION 9:



#### ITERATION 10:

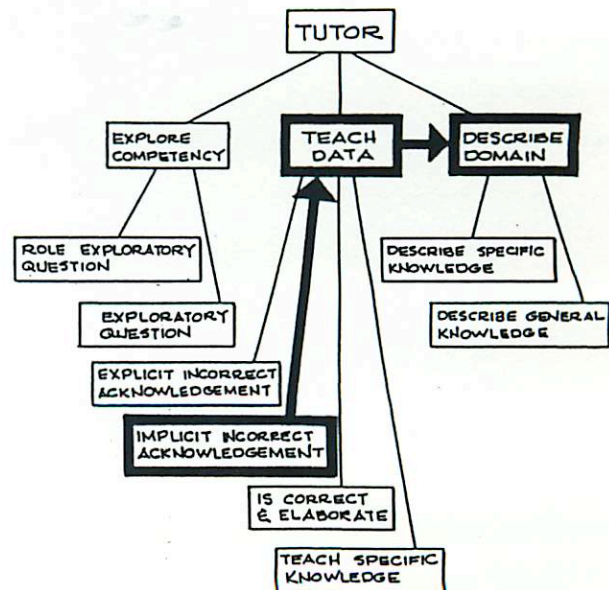
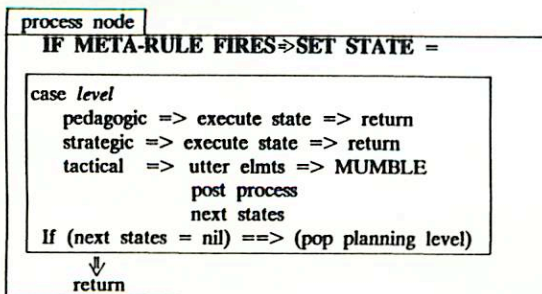


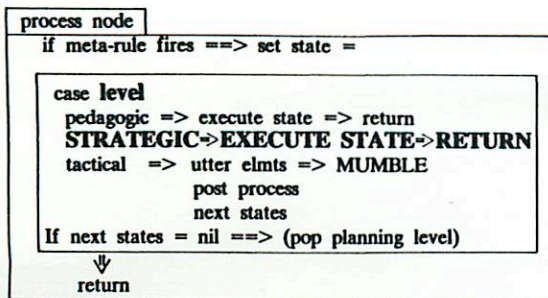
Figure 36: Implicit to Describe Snapshot.

Once here a meta-rule, S3-DESCRIBE, moves the tutor to *describe-domain*, terminating

the question-answer approach and beginning a lengthy descriptive passage about a single topic. This happened because the threshold of wrong answers has been met (as recorded by a register) and there is a link from the major topic, "climate in Washington and Oregon," to an undiscussed geographical factor on which it is dependent, namely the "Japan Current."

From *describe-domain*, the tutor takes the default path to *describe-specific-knowledge* at the tactical level, Figure 37

ITERATION 11:



ITERATION 12:

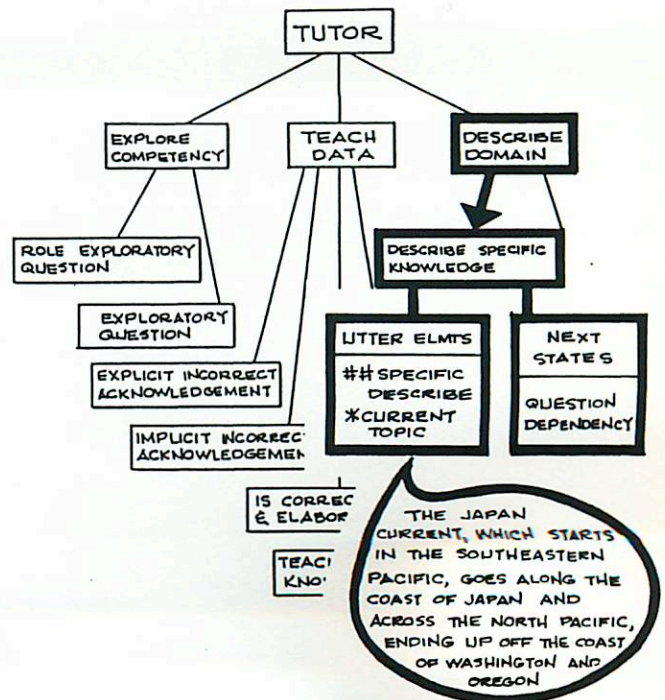
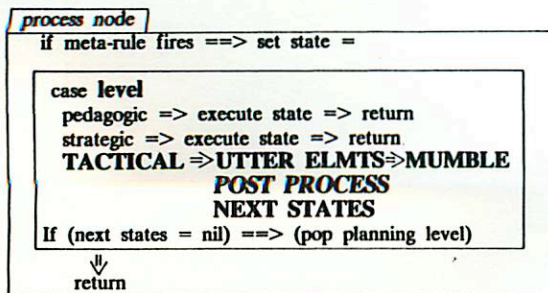


Figure 37: Describe Strategy Snapshot.

and constructs an utterance from the specification *specific-describe (current-topic)*. *Specific-describe* enunciates each attribute of its evaluated argument and *Meno-tutor* says

"the Japan Current, which starts in the Southeast Pacific, goes along the coast of Japan and across the North Pacific, ending up off the coast of Washington and Oregon."

This brief traversal through the network and algorithm is meant to illustrate the interaction of meta-rules and data structures during the planning and generation of utterances. A brief discussion was provided on the process of selecting the rules and a way to increase the sophistication of the existing heuristics to select these rules, e.g., by rewriting the function `execute-state`.

#### 4.4 Discussion of the rules

Meta-rules handle the high level shifts in discourse: they generate the transitions that are responsible for producing both flexibility and continuity in the discourse. The existence of these rules make implicit claims about the necessity of rules in a tutoring system. In this section I begin a discussion about the design of these rules, the way they are edited and some of the assumptions that lie behind their development. In Chapter 6 I discuss the theoretical need for such rules.

##### 4.4.1 Mechanisms for Authoring rules.

The meta-rules evolved through a fairly empirical process. The tutoring program was run many times to generate specifications for text. As insufficiencies in the Preconditions of the rules were noted, the rules were "fine-tuned" and retested until the specifications became better defined.

The author of the system has two mechanisms available for defining a rule and for specifying when it will run. One is to use predicates in the Precondition slot of the rule definition to prohibit the rule from being considered except after due deliberation. For example, T6-A.IMPLICITLY has a two predicates which require evidence that the student is in doubt and that the strategy is ineffective before it will run. Both these judgements are global and require inference about the pattern and structure of the discourse thus far. In a sense, the Preconditions constrain the rule to run only if the system can make an inference both about the "state" of the student's knowledge, or confusion, in this case and the "state" of the discourse.

The other mechanism for controlling a rule is through indication of Prior State, the state during which a meta-rule can run. The Prior State arbitrates when (within which state) a rule can be enabled. The combination of Prior State and Preconditions takes into account all pedagogical and rhetorical considerations available to the system. The two mechanisms act in concert to define when and under what circumstances a meta-rule will be triggered.

Although the rules are independent, they interact slightly through this preemption process. Thus when a rule like S3-DESCRIBE runs it effectively relegates all other strategic rules to an inactive mode. These other rules might never reemerge if the



running of the rule places the discourse in a mode which forever preempts their consideration. Thus, for instance, rules which distinguish among *teaching* and *exploring* strategies are made null and void when a rule moves the system into the *describe* strategy.

#### 4.4.2 Editing a rule.

As an illustration of how the slots of meta-rules effect discourse structure, I describe an instance in which a change in preconditions would have altered a discourse. The example discourse is the same as that used in the previous section and begins after one question has been answered incorrectly and little is known about the student's knowledge. During the iteration of process-node, the function preempt-path is called to test the meta-rules; if a meta-rule is triggered current state will be preempted and one of two available strategies, *explore competency* or *describe domain*, will be selected. Two meta-rules might be enabled at this time, S1-EXPLORE and S3-DESCRIBE; the former moves the tutor to question the student and to assess his knowledge and the latter moves the tutor to provide a lengthy description of information from the knowledge base.

However, neither meta-rule was enabled in the original example because the present implementation defines teaching as an exchange in which one topic is clearly explained before the tutor moves on to a new topic. In accordance with this philosophy the Preconditions of the meta-rule have been written to require that the register *\*topics\_complete* be satisfied before the rule can be enabled.

This approach is admittedly restrictive; for instance, it will not permit exploration of even a second topics to define a student's knowledge until each question is correctly explained. This rule in fact might lead to arbitrarily tiresome discourses.

However, meta-rules can be modified. Any of the four parts of a meta-rule, Prior State, Precondition, Action, or Postprocessor, can be changed. For instance, in the case of the example above, editing the rules S1-EXPLORE and S3-DESCRIBE to eliminate the Precondition test on *\*topic\_complete* would have changed the given discourse to move ahead to question a series of new topics (*explore competency*) or to describe a single topic in depth (*describe domain*) without first correcting the student.

## 4.5 Summary

In this chapter, I have described the data structures and algorithms used to generate tutoring discourse in Meno-tutor system. I have not provided details of the theory of tutoring nor shown extensive results. Both are described elsewhere (Chapters 5 and 6). However, the important issues related to building a tutoring component are

described above and summarized below.

A generic ATN environment, with the addition of hand-tailored meta-rules, was built to generate flexible and responsive output in the face of student failure. The system is non-domain and non-student specific and serves to demonstrate the reasonability of separating tutoring knowledge from domain and student knowledge. The design of the meta-rules provides a mechanism for making global considerations in discourse planning and incorporates some of the intuitions and complexity of discourse management which might otherwise fall under the rubric of linguistic reasoning.

This research on discourse control processes is applicable to any intelligent interface which must reason about the user's knowledge instead of providing him with directly retrievable data. This includes explanation systems, discourse systems and text generation systems, in addition to tutoring systems.

## CHAPTER V

### RESULTS AND EXAMPLES

This dissertation has recounted the research behind the development of a tutor based on an explicit network representation of tutoring states. In a sense, the tutor has the ability to view the discourse and the student's ability from many perspectives. It can shift between topics and discourse approaches, at times retrieving data, at other times questioning the student superficially about a number of topics, or probing him in depth about a single topic. As a result of this diversity in discourse generation, the tutor appears to have a deeper understanding of the student, as exhibited by the range of utterances or viewpoints it can generate. This chapter presents some experiments made with Meno-tutor to test this range of responsiveness. The experiments demonstrate the system's flexibility in working with a range of different students including one which is knowledgeable, another which is confused, and one with a misconception.

In each example, the production of a discourse involved straightforward modification of system parameters, primarily the meta-rules, the expert knowledge base, and Michelin Ratings. The first two examples were actually generated by the original Meno-tutor with the modifications as described. Traces of these runs can be found in Appendix A. The third and fourth discourses were not implemented; yet the necessary modifications are clearly presented and seem to include simple extensions to the system.

Section 5.1 presents a discussion of the variety of utterances available from the system in its original form. Sections 5.2-5.4 show the results of changing three system parameters, meta-rules, knowledge base, and Michelin Ratings. The next section, Section 5.5, shows the result of working with a student who is knowledgeable about the subject area and Section 5.6 presents a summary of the chapter.

#### 5.1 The Variety of Utterances

There are two broad areas of linguistic control in Meno-tutor that roughly correspond to the three levels of the Discourse Management Network ("DMN"). The top level of the DMN is responsible for maintaining *context* of the discourse or handling *how* the utterances flow together. The bottom two levels handle the *content* of the utterances or *what* is said. The two areas provide an artifact for teasing apart an utterance and use its constituent parts to plan and generate text.

The first area, corresponding to the top level of the component, is sensitive to high-level rhetorical concerns, such as maintaining the questioning paradigm or discovering an underlying misconception. States at this level, for example, establish a new topic to be probed, but leave the actual questioning to the lower states.

The second area of control, corresponding to the lower two levels, is data driven and sensitive to the complexity of the knowledge base and the evolving student model. States at this level determine, for instance, which topics will be discussed and which data will be used first in the tutor's utterances. The work of these two areas provides a variety of rhetorical as will be discussed below.

The next two figures present 12 base-level utterances produced by the tutor from a single tutoring gesture and without any modification of the system. The utterances are meant only to suggest the variety possible through the DMN; these utterances are not the only ones available. The actual available number is dependent upon the number of paths through the 40 states. The utterances in the figure represent a variation of only a half dozen states at the lowest level.

The bottom two levels of the DMN are the most significant in extending the possibilities for textual variety. Figures 38 and 39 present 12 utterances each determined by the single Pedagogical State *tutor* which, in turn, determines the goal of the utterance. Thus the goal of each is to tutor the student by presenting data from the domain, either through explanations, examples, or questions. If, on the other hand, the utterances were generated another tutoring gesture, say *hack*, the goal would be to probe the student's misconception, by proposing, verifying and repairing a misconception. The topics focussed on is either "climate in Washington and Oregon" or "rainfall in Washington and Oregon." These two topics and the related one, "rainforests", are located in the original expert knowledge base in Chapter 3, Figure 15.

The first utterances in line 1 of both figures was produced using the single Pedagogical *tutor* and the Strategy *explore competency*. Both utterances are directed at exploring and defining the student's threshold of knowledge; each questions the student's model of the specific topic. Utterance 2 in Figure 39 is built from a different Strategy, *describe domain* and its goal is to describe a topic rather than exploring it. The remaining utterances in each figure are constructed, again, from the single Pedagogical State *tutor* and from a new Strategy, *teach data*. Their goal is to tutor by either questioning or explaining a piece of data related to the current topic. The explicit formulation of either the question or the explanation is achieved by the particular choice of Tactical State at the bottom level.

Lines 3, of both figures, are negative acknowledgements of an incorrect answer and an accompanying explanation; the utterances recognize the wrong answer and complete the discussion by providing the correct answer. The responses in lines 4 acknowledge the correct answer while elaborating upon it. Line 5 of Figure 38, makes an implicit utterance about the student's incorrect answer and moves on to the next topic. Its goal is to dismiss the topic, keeping it in abeyance, possibly for a later discussion. Both lines 6 respond to the student's incorrect answer by making a succinct explanation of the correct answer. Line 7 demonstrates the utterance of a question which requires an inference about a prior topic.

**TOPIC: Climate in Washington and Oregon**

1. "What do you think about the climate there?"
 

|           |                      |
|-----------|----------------------|
| Strategy: | Explore              |
| Tactic:   | Exploratory question |
  
3. "No, it is warm in Washington and Oregon."
 

|           |                                    |
|-----------|------------------------------------|
| Strategy: | Teach Data                         |
| Tactic:   | Explicit Incorrect Acknowledgement |
  
4. "Right, it is indeed warm there."
 

|           |                          |
|-----------|--------------------------|
| Strategy: | Teach Data               |
| Tactic:   | Is Correct and Elaborate |
  
5. "Well, [... move to another subject] "
 

|           |                                    |
|-----------|------------------------------------|
| Strategy: | Teach Data                         |
| Tactic:   | Implicit Incorrect Acknowledgement |
  
6. "It is rather mild there."
 

|           |                          |
|-----------|--------------------------|
| Strategy: | Teach Data               |
| Tactic:   | Teach Specific Knowledge |
  
7. "What does that tell you about the temperature of the Japan Current?"
 

|           |                             |
|-----------|-----------------------------|
| Strategy: | Teach Data                  |
| Tactic:   | Question Reverse Dependency |

**Figure 38: Utterances to Teach about Climate.**



**TOPIC: Rainfall in Washington and Oregon**

2. "What do you think about the rainfall there?"  
 Strategy: Explore  
 Tactic: Exploratory question
3. "There are rainforests in both Washington and Oregon"  
 Strategy: Teach Data  
 Tactic: Describe Specific Knowledge
4. "What does that tell you about the rainfall in Washington and Oregon?"  
 Strategy: Teach Data  
 Tactic: Question Reflexive Factor
6. "No, it is wet in Washington and Oregon."  
 Strategy: Teach Data  
 Tactic: Explicit Correct Acknowledgement
8. "Right, it is indeed wet there."  
 Strategy: Teach Data  
 Tactic: Is Correct and Elaborate
11. "It is rather wet there."  
 Strategy: Teach Data  
 Tactic: Teach Specific Knowledge

**Figure 39: Utterances to Teach about Rainfall.**

The range of utterances produced thus far is a measure of the sophistication of the tutor and reflects its superficial understanding of the student and his problems. In the discourse examples presented in the remainder of this chapter, the base level utterances are extended through the modification of system parameters.

## 5.2 Discourses Produced by Extending Meno-tutor

### 5.2.1 Varying Meta-Rules.

The discourse in Figure 40 illustrates how meta-rules affect discourse production. This discourse was generated by the original Meno-tutor with the modifications described below. A partial trace can be found in Appendix A.

The meta-rules implement the high level structural shifts in a discourse and respond to the variations in the model of the student. The rules determine, for instance, how strongly, and with what techniques, the tutor will react to the student and how much effort, in the form of probes or questions, the tutor will use in a particular situation, such as the one presented by the student who lacks enough knowledge to continue the discourse.

The discourse in Figure 40 covers the same topics as did the original discourse about rainfall. However, instead of exploring the student's frontier of the knowledge, as did the original discourse, three changes in the meta-rules caused the tutor to first probe a possible misconception (see Section 4.4 for a discussion of how to edit the meta-rules).

The first meta-rule to be modified was S6-JETTISON. This rule moves the tutor up to the previous planning level after sufficient evidence is presented that a strategy has failed to work with the student or confidence in the student's ability has been established. The rule establishes a conservative measure on a test of teaching effectiveness; it is not invoked until the register \*terminate\_strategy is set implying that: 1) all the topics have been discussed, or \*topics\_complete is true, and 2) the model of student knowledge is well established, or \*competence\_confirmed is true. This rule is typically triggered at a turning point in the conversation, such as after several topics are explored or the tutor has confidence in the student's knowledge.

Writing the preconditions in this way so that the meta-rule is triggered after only a single wrong answer, shifts the path of the discourse to abruptly probe a misconception after the first incorrect answer.

The second meta-rule to be modified was P2-HACK. This rule moves the tutor to probe potential misconception. Typically, it is triggered when all topics are complete, i.e., either answered correctly or corrected by the tutor. The preconditions for this rule are three tests: \*misconception\_triggered, \*topics\_complete and \*completion\_acknowledged.

---

*Tutor:* Do you think it rains much in Oregon?

*Student:* No.

*Tutor:* Why do you think it doesn't rain much in Oregon?

*Student:* Because the neighboring states are dry.

*Tutor:* The neighboring states are dry, but, you can not generalize from that. In fact, it does rain a lot in Oregon. Can you guess what causes the rain there?

*Student:* Well, there are high mountains in the region and the states border the Pacific Ocean.

*Tutor:* Yes. The Pacific borders Oregon. How do you think it is involved in the heavy rainfall there?

*Student:* Ocean air rises and cools upon reaching the land.

*Tutor:* That is right.

---

**Figure 40: The Second Rainfall Discourse.**

---

The first is set when a student's response is linked to a misconception in the expert knowledge base. \*Topics\_complete is toggled when new topics presented by the tutor have been either answered correctly by the student or corrected by the tutor. \*Completion\_acknowledged is toggled when the tutor indicates that the current topic is finished.

By changing this meta-rule so that it triggered after only \*misconception\_triggered, the thrust of the discourse was shifted to *hack* misconception. In this example, the student's explanation of his reasoning, immediately after the first misconception probe, did not trigger a known misconception, i.e., the suspicion of a misconception could not be verified. In the next line, the tutor provided the correct answer and moved on to

essentially continue the questioning paradigm established by the first question.

The third item to be modified was to the order of evaluation of the meta-rules. In the present design, several rules may be enabled at once; only one will be triggered and run. The selection of a rule is made by taking the first element of a list of enabled rules. The order of evaluation of the rules therefore determines the order of the enablement of a rule. This order was originally the inverse of the popularity rule: i.e., the least popular rule was tested first. If the most popular rule were tested first, it would be triggered frequently, effectively blocking out the least popular rule.

Thus S6-JETTISON, which had been at the end of the list of strategic rules, was placed first in the list to produce the required result. A great deal of experience is needed before we can understand the effect of these rules either on each other or on discourse generation. Experiments such as the ones described here will enable us to refine these rules and understand their effect on discourse.

### **5.2.2 Varying the Expert Knowledge Base.**

A major goal of this research has been to demonstrate the power of isolating the tutoring component of a discourse system. One reason for this goal was to see if the tutoring component could be interfaced with a different expert knowledge base and a different language generator and could then teach a new subject and even "speak" in a new language. According to this proposition, for instance, a Pascal knowledge base and, say a Chinese language generator, could be interfaced with the tutoring component and the resulting system could interrogate a student in Chinese and teach him about programming in Pascal. No changes in the tutoring component would be required to accommodate either the different domain or the different output language.

In this example, I forgo the opportunity to produce a discourse in Chinese. Instead, I have produced a tutoring discourse about Pascal using the original Meno-tutor and a different expert knowledge base as described below. Again this discourse was actually generated by the original Meno-tutor with the change in expert knowledge base. A trace of Meno-tutor's operations during the production of this discourse is given in Appendix A.

Meno-tutor was originally developed as part of a larger research effort directed at building an on-line run-time support system for novice Pascal users [Soloway et al., 1983; Soloway et al., 1981]. As a part of this effort, a Bug Finder was developed which detected run-time semantic errors in novice Pascal programs [Soloway et al, 1981; Rubin, 1981]. The Bug Finder was in use for 4 semesters on classes of several hundred students at the University of Massachusetts. It detected simple errors in Pascal looping programs. It then passed messages to the tutor which could interpret information about the type of error, from within a variety of categories, and the line numbers of related variables. and would interact with the student around the error.

```

1  PROGRAM LESSON1(INPUT, OUTPUT);
2  VAR
3  SUM, GRADES, STUDENTS: INTEGER;
4  MEDIAN: REAL;
5  BEGIN
6  SUM:=0;
7  STUDENTS:=0;
8  READ(GRADES);
9  WHILE GRADES>0 DO
10 BEGIN
11 SUM:=SUM + GRADES;
12 STUDENTS:=STUDENTS+1;
13 GRADES:=GRADES+1;
                                should be READ(GRADES);
14 END;
15 MEDIAN:=SUM/STUDENTS;
16 WRITELN
17 ('THE MEDIAN GRADE IS', MEDIAN:3)
18 END.

```

**Figure 41: A Student Pascal Program.**

---

The program in Figure 41 was an actual program submitted by a novice programmer. The dialogue in Figure 42 was generated by Meno-tutor after the Bug Finder was retired. It is based on simulated messages from the Bug Finder and simulated student input.

A portion of the Pascal is represented in Figure 43. It was encoded in the same kind of KI-ONE semantic network as was the first discourse in Chapter 3, Section 3.2. Thus the Pascal knowledge base was available for interpretation by the same techniques as used in the earlier discourse. However, this knowledge base was richer than the rainfall knowledge base because it contained assumptions about incorrect, yet commonly held, programming misconceptions. The hierarchical organization of topics was based on extensive cognitive studies about programming plans used by novices to represent Pascal knowledge and variations on these plans were associated with misconceptions [Bonar, 1984].

Given the program of Figure 41 and what the student got right, the tutor inferred that the student understood the basics of loop programming. To be certain that the tutor and the student shared a common vocabulary, the first two questions tested general looping concepts (lines 1 and 2). These two answers were correct, suggesting that the student understood the rudiments of looping constructs and the role of the control variable.

*Tutor:* Do you know that GRADES, in line 9, is a control variable for the WHILE loop?

*Student:* Yes

*Tutor:* Does that mean that GRADES should be  
a) Initiated before the loop?  
b) Added into SUM within the loop?  
c) Modified within the loop?  
d) Both a) and c)

*Student:* C

*Tutor:* You seem to understand about the control variable of the WHILE loop. Now let's talk about some other command statements. What do you think READ (GRADES) in line 8 does with the value of GRADES?  
a) Reads all values of GRADES at one time.  
b) Reads in a single value of GRADES.  
c) Assigns several new input values to GRADES.

*Student:* A

*Tutor:* Then do you think that GRADES ::= GRADES + 1 updates the next of those values?

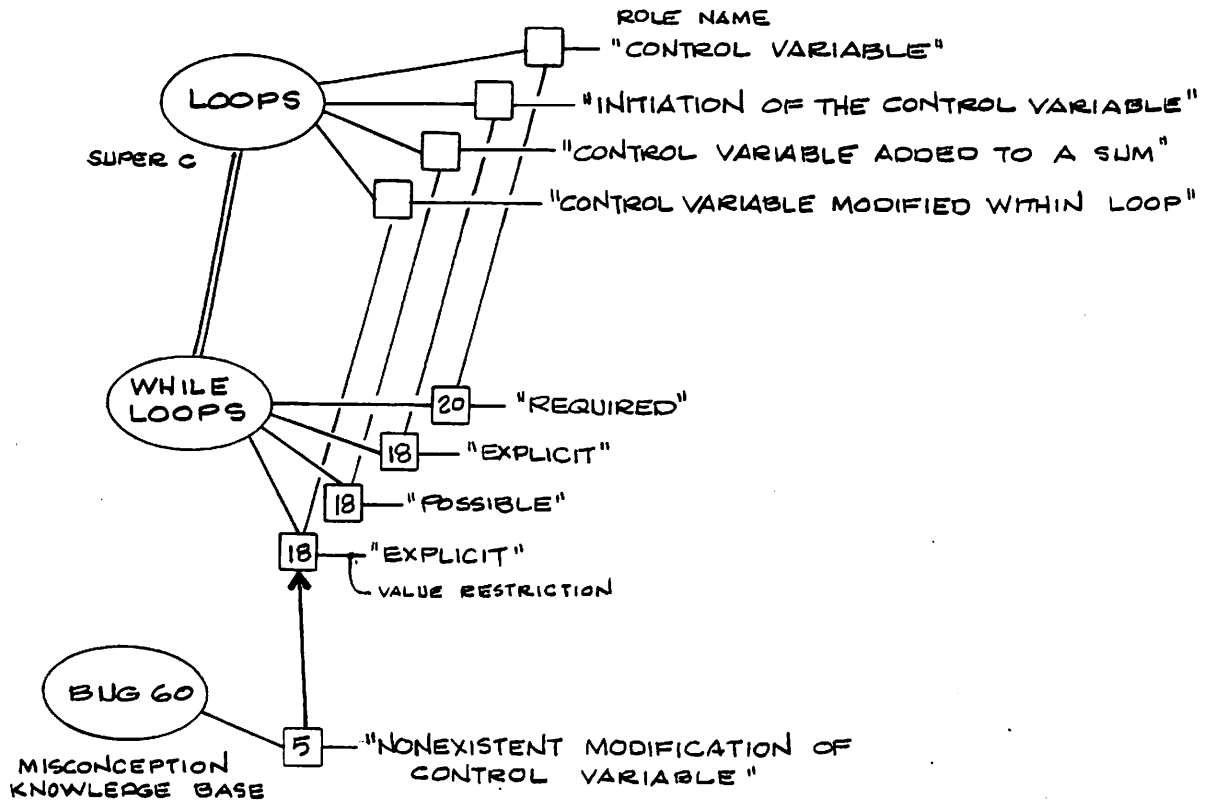
*Student:* Yes

*Tutor:* There is your misconception. Adding one to a variable is a common way to step through a range of values. But it cannot apply here because your READ instruction in line 8 retrieves only a single value. You should replace line 13 with a second call to READ (GRADES).

**Figure 42: Meno-tutor Discussing the Student's Errors.**

---

In line 3, Meno-tutor analyzed deeper programming concepts. Psychological studies identified the explicit error in the student's program with several misconceptions which have been labeled "READ-loop bugs". The syntactic error seems to have stemmed from either of two misconceptions: a belief that (1) a single call to READ will access all the values of a variable at once, or (2) updating the control variable (line 13) will do a tacit READ through each value of the variable each time through the loop. The questions in lines 3 and 4 are an attempt to discriminate between these two misconceptions and since both answers were wrong indicate that the student had both misconceptions.



**Figure 43: The Expert Knowledge Base for the Pascal Discourse.**

The tutor's response in line 5 is an example of the "grain of truth" speech pattern (see Section 3.1). It confirms and reinforces the student's correct knowledge about using the ADD 1 operation to bump up the control variable, while pointing out that it is inappropriate to use this construct for the control variable inside the WHILE loop.

### 5.2.3 Varying Michelin Ratings.

The next example extends the variety of possible discourses by changing Michelin Ratings on the expert knowledge base. Michelin Ratings have the effect of encoding an importance hierarchy onto the data used in the discourse (see Section 3.2) and provide a way to quickly retrieve topics, ordered by teaching or learning priority. This numerical annotation metaphorically provides a way to pull topics out of the knowledge base

ordered by importance. Figure 44 suggests a new ordering on the topics of the original rainfall data base whose use will be described below. The discourse was not actually generated by Meno-tutor, but the changes appear to be straight forward.

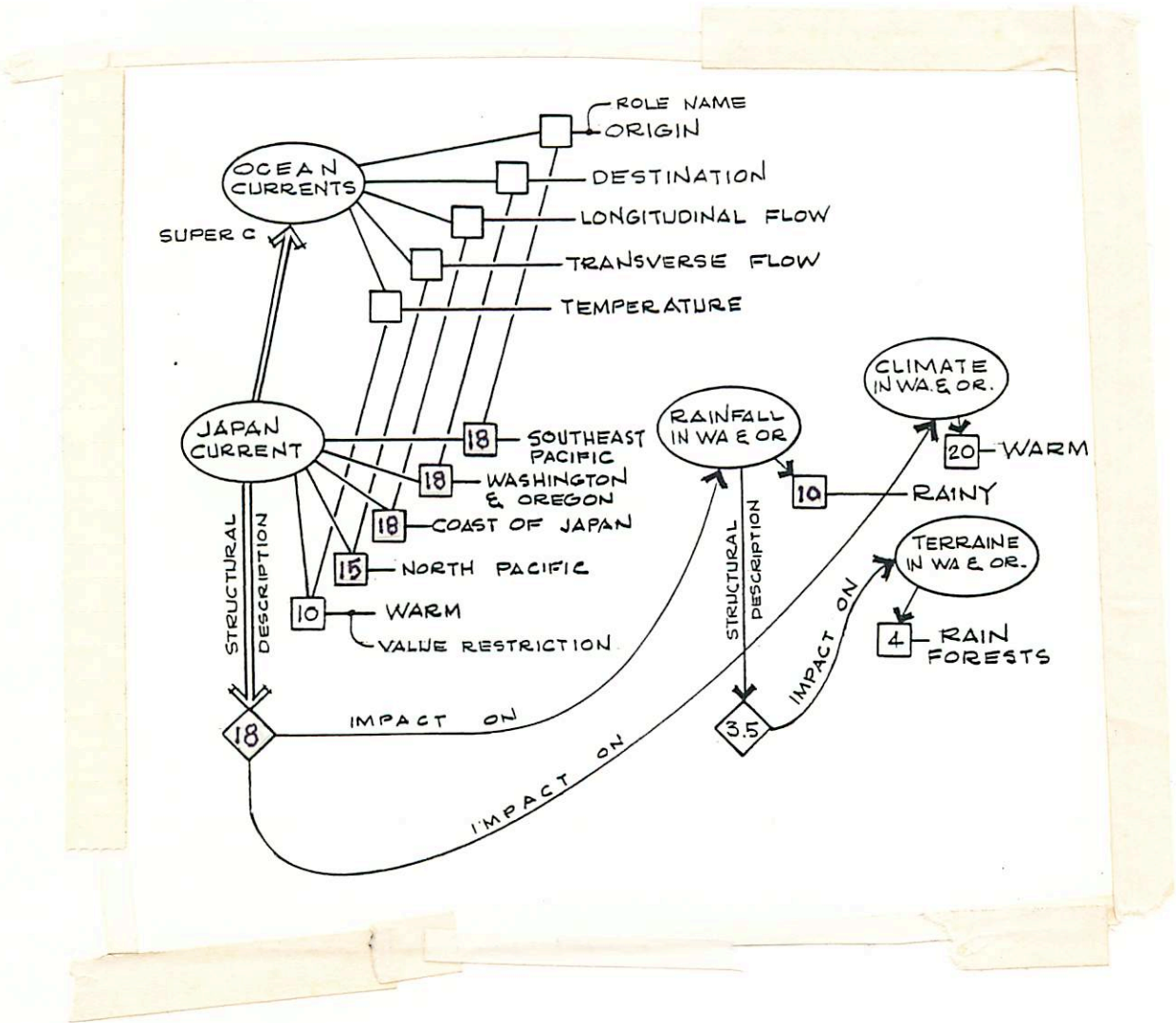


Figure 44: The Rainfall Knowledge Base with new Michelin Rating.

The discourse in Figure 45 shows the result of using the annotation rating suggested by Figure 44 on the original knowledge base. Since the new Michelin Ratings highlight the concept of "Japan Current" over the concepts of "warm climate" or "heavy rainfall," the roles of the "Japan Current" are given priority over the general descriptors. As the student continued to give correct answers to the questions about the Japan



Current new role topics were selected for questioning.

---

**TUTOR:** Do you know what the climate is like in Washington and Oregon?

**STUDENT:** Is it cold?

**TUTOR:** No, it is rather mild. Can you guess about the effect of the Japan Current on the climate there?.

**STUDENT:** I don't know.

**TUTOR:** Well, the Japan Current begins in south eastern Pacific. Can you guess in which direction it travels?

**STUDENT:** Northwardly?

**TUTOR:** That is right. Do you know where it ends up?

**STUDENT:** Near Washington and Oregon?

**TUTOR:** Good. How do you think that current affects the current in Washington and Oregon?

**STUDENT:** It's probably rainy.

**TUTOR:** It is indeed. There are rain forests in both Washington and Oregon. What does that tell you about the temperature of the Japan current?

**STUDENT:** It's warm.

**Figure 45: Discourse Produced by Varying Michelin Rating.**

---

The Michelin Rating is supplied by the author of the tutoring system based on his teaching experience. Extensions of Meno-tutor into new domains suggests that new authors will place their own ideas about topic importance on the new knowledge base. Ideally, such new orderings and Michelin Ratings should be based on empirical evidence.

of learning and teaching in the domain. Building a new knowledge base should enforce the chunking paradigm in knowledge representation and acquisition (see Section 2.4.1). Clearly the knowledge base should include and should represent the links between topics whose successors, or predecessors are clearly indicated. There are only a few domains in which in-depth empirical data about such chunking and relations exist e.g., mathematics, chess, programming, physics (see Section 2.4.1). Thus, there are only a few domains in which the system's knowledge of the domain can actually be based on empirical studies of learning in that domain. Until such empirical studies of learning are performed in each domain, tutoring systems in these domains reflect only the author's personal choice of topic importance and topic relatedness. Without the empirical basis to tutoring systems, one author's choice of important topics for his tutoring system might be another author's list of trivia topics for the same domain.

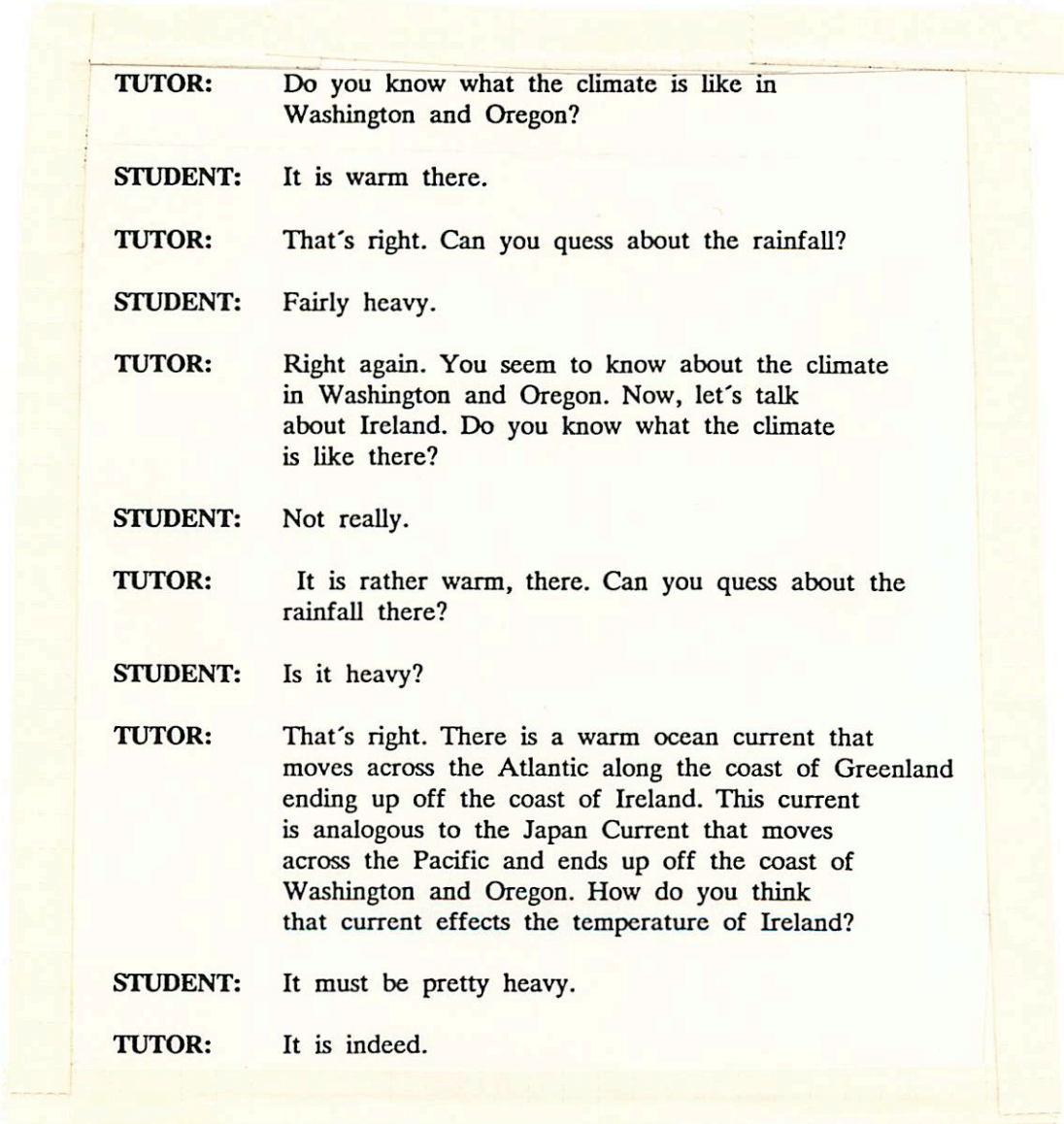
#### 5.2.4 Working with the Knowledgeable Student.

The next discourse suggests an example of how the system would behave with a knowledgeable student. The discourse was not implemented, yet the required changes are straightforward refinements to the existing system. The discourse in Figure 46 shows how, beginning with the same goal and topics as the original rainfall discourse, the system eliminated detailed discussion of *rainfall in Washington and Oregon* and talked instead about an analogous topic, *climate in Ireland* because the student's answers to the first two questions indicated that he knew the topic well.

The decision to discuss Ireland instead of Washington and Oregon was made because the goal of tutoring includes clarification of a body of knowledge to which the student has already become familiar. Once the tutor determines that the student already knows a body of knowledge and requires no further help with it, there is no reason to dwell on that body of knowledge. Therefore, the meta-rule P3-COMPLETE fires and the system is moved to discuss an analogous topic, completing the current topic, before it does.

In this example, the rule P3-COMPLETE was triggered because two registers were set: `*topics_complete` and `*competence_confirmed`. The first was set after all topics introduced by the tutor were either satisfactorily answered or completely corrected by the tutor; the second was set when `#topics_list` and `#topics_questioned`, both entries on the property list of the variable `*counters`, became equal.

There are two motives behind activation of the P3-COMPLETE rule. The first is to extend the breadth of the tutor's understanding of the student's knowledge. When this rule is triggered the system furthers its efforts to learn more about the student's knowledge in other domains. The second motive for this rule is to enable the tutor to use an analogous domain in order to reveal the student's errors and misconceptions, if any. The existence of errors and misconceptions provides a window into the structure of the student's knowledge which allows the tutor to probe the student's knowledge.



**TUTOR:** Do you know what the climate is like in Washington and Oregon?

**STUDENT:** It is warm there.

**TUTOR:** That's right. Can you guess about the rainfall?

**STUDENT:** Fairly heavy.

**TUTOR:** Right again. You seem to know about the climate in Washington and Oregon. Now, let's talk about Ireland. Do you know what the climate is like there?

**STUDENT:** Not really.

**TUTOR:** It is rather warm, there. Can you guess about the rainfall there?

**STUDENT:** Is it heavy?

**TUTOR:** That's right. There is a warm ocean current that moves across the Atlantic along the coast of Greenland ending up off the coast of Ireland. This current is analogous to the Japan Current that moves across the Pacific and ends up off the coast of Washington and Oregon. How do you think that current effects the temperature of Ireland?

**STUDENT:** It must be pretty heavy.

**TUTOR:** It is indeed.

**Figure 46: Discourse with the Knowledgeable Student.**

---

### 5.3 Summary

This section has pointed up the need for a tutoring system to respond to a student in a custom-tailored way which reflects his level of knowledge. The examples illustrate the variety in the speech patterns and discourse approach that is required to achieve such custom-tailored discourses. Meno-tutor allows variety in its discourse primarily through two sources: the variable paths through the DMN and the modification of the separate modules of the system, e.g., the expert knowledge base and the teaching strategies (meta-rules).

This chapter demonstrated the result of making three modifications to the system to expand its discourse generation ability beyond the possibilities provided levels of the DMN. The three system modifications were: vary the meta-rules, change the Michelin Ratings and switch expert knowledge bases. These modifications produced discourses which could handle the person who had a misconception, another whose misconception could be repaired, and yet another who was too knowledgeable for the discourse to continue along the preplanned path.

## CHAPTER VI

### CONCLUSIONS

This thesis proposed a computational theory of tutoring and supplied a description of a specific implementation of that theory. The intention was to identify some of the primitives of discourse generation that relate to tutoring. The thesis has both theoretical and practical implications. Theoretically, the study (1) established some minimal conditions required for tutoring text generation and (2) demonstrated a method for separating the tutoring component from the linguistic or domain knowledge of a teaching system. Practical implications include providing evidence that an explanation system and an intelligent interface must respond at the level of understanding of the user. Other implications are for education and are manifest both in the potential of new instruments in the classroom and in the advent of technologies for teaching learning.

The conclusions of this study, along with some of the claims and justification of the theory of tutoring, are presented in this chapter. Some extensions to the system and future research are also described. In Section 6.1 the goals are reiterated and an attempt made to assess my success in achieving them. Section 6.2 presents the components of the theory of tutoring and includes some justification for each. In Sections 6.3, 6.4, and 6.5 the features, limitations, and implications of the system are presented. Section 6.6 explores the implications of this research to education and Section 6.7 discusses further research directions.

#### 6.1 A Review of the Goals

Meno-tutor was designed to demonstrate that a combination of a knowledge representation and a specific system architecture can make decisions about tutoring. The system does not advocate a particular approach to tutoring, but rather suggests a way to direct the system's output to the student's level of understanding. Meno-tutor uses parameterized states and rules, which are exceptionally highly structured, and an unmarked control flow that is preemptable based on inferences made about discourse and student history.

The mechanisms posed in this research are not final; additional rules, more complex states, and more intricate control structures would contribute to the richness of the system's tutoring capabilities. However, the intent has been to discover a sufficient set of machinery and knowledge necessary to carry on acceptable tutoring discourse and to elucidate this tutoring process.

## 6.2 An A.I. Theory of Tutoring

I take the position that the behavior of the system as shown in the examples of Section 5.1 corresponds to human tutoring behavior and can be used to validate claims about the power of its machinery. That is, I substantiate the A.I. theory by using my system as an example of how human-like tutoring behavior can be produced from a machine. Strong claims about the mechanisms are made in order to initiate a dialogue about the effect of system design on the generation of discourse. In the following sections, I list the components of the A.I. theory of tutoring discourse that have emerged during the course of this research.

### 6.2.1 Tutoring as a Function of Language Generation.

The theory of tutoring proposed by Meno-tutor is dependent upon the of its language generation facility; it adapts what is said to the context of student and discourse history. The program uses knowledge about tutoring strategies, complex communication skills, and the level of the student's knowledge to generate reasonable tutoring discourse.

I designated such a system "context-dependent" and contrasted it with "retrieval-oriented" systems; in the latter systems the goal is to retrieve the correct answer and the input/output routines act only as an interface to the knowledge retrieval system. The context-dependent nature of Meno-tutor distinguishes it from other tutoring systems, which are oriented toward information retrieval, for example, WHY [Stevens & Collins, 1977], BIP [Barr et al., 1976], and GUIDON [Clancey, 1979] (see Chapter 2). Other systems have been designed in two stages: (1) a knowledge retrieval system, and (2) a natural language "front end interface." In these latter systems the *interface problem* is considered secondary to the *retrieval problem*, and the two subsystems are designed as if they were theoretically independent. Such a design – typical of many teaching and explanation systems – considers the form of the response secondary to its "correctness".

In Meno-tutor the problem of knowledge retrieval is considered secondary to the problem of understanding and formulating an appropriate response to the student. Its primary goal is to make some speculations about the student's intent or problem and to modify its default response into one that is consistent with its interpretation of the student's knowledge. The system's ability to generate a variety of alternative utterances according to the discourse context and the tutor's assessments (see Section 5.1) is critical to this process.

The design of Meno-tutor constitutes a major departure from the information retrieval viewpoint of tutoring where natural language is seen merely as a front end. In Meno-tutor, the actions of the language planning component are central to determining its response. In this respect, the design of Meno-tutor is similar to Lehnert's conceptual theory of questioning and answering [Lehnert, 1977]. In Lehnert's system the user's

questions are parsed and the answers “framed” according to conceptual dependency categories which model the intent of the user more than they model the syntax of either the question or the answer. In a similar way, the decisions made by the tutoring component are constructed from a model of a generalized response that is constrained by an adopted frame and limited to speech patterns available within that frame. The model of a generalized frame is updated and refined to the individual student, but is still a variation from a category of responses.

In sum, Meno-tutor is based upon a theory of tutoring which is centered around its language generation facility. The theory, which takes into account rhetorical patterns, stylistic considerations, and discourse rules, is centered on language generation.

### 6.2.2 Requirements for Style.

I claim that rhetorical and pedagogical style are an integral component of tutoring discourse. By style I include, for instance, a decision made about expository style or topic selection. That some style is necessary is made clear by virtue of the fact that the contrapositive is false: random changes in expository style – or worse yet, no change at all – are not effective in engaging the student. The example of a bad dialogue in Figure 47 displays a discourse about the same topics used in the discourse of Figure 1, but with no shifts in expository style beyond the familiar question-and-answer approach. The dialogue is coarse and unyielding: it shows no recognition that the student has no correct answers and it continues to badger him about topics of decreasing importance after he fails to correctly answer questions about the more significant topics. This example offers evidence that guidelines for rhetorical and pedagogical style are needed.

Though the claim that style is important for tutoring is an intuitive one and not backed up by psychological studies, there are studies of human tutoring which suggest that a variety of approaches and styles are used by expert tutors to manage tutoring discourse (e.g., Collins & Stevens, [1979]; Collins et al., [1974]). This research presents evidence that tutors are guided by long-term goals such as those listed in Figure 48.

Each goal is implemented by the human tutor who uses rules to maintain discourse relevance and selects speech acts from within a set of fairly predictable speech patterns (see Section 3.1). Handling each of these activities suggests that tutoring is a planned and structured activity that requires pedagogical and rhetorical style.

Computationally, a reasonable solution to this need for style might be to use a grammar or decision tree to guide movement among alternative pedagogical approaches. This has been done (see, for instance, [Barr, 1976]). Meno-tutor’s approach, however, is to determine a pedagogic style through preemption based on context, i.e., information from the student model or the discourse rules is allowed to preempt the default discourse structure. In sum, the model of style presented here presumes that rhetorical and pedagogical style are integral components of tutoring discourse.



*Tutor:* Do you know what the climate is like in Washington and Oregon?

*Student:* Is it cold?

*Tutor:* No, it is rather mild. Can you guess about the rainfall there?

*Student:* Normal, I guess.

*Tutor:* No, it is rather heavy. Do you know where the Pacific Current starts?

*Student:* No.

*Tutor:* It starts along the Southeast Pacific. Do you know in which direction it travels?

*Student:* Southerly?

*Tutor:* No, it travels across the North Pacific. Do you know where it terminates?

*Student:* No.

*Tutor:* It ends up along the coast of Washington and Oregon. How do you think that current affects the climate

**Figure 47: A Bad Discourse.**

Refine the student's procedures.  
 Correct errors before omissions.  
 Concentrate on one part of the causal structure.  
 Ask for a rule in special instances.  
 Provide a hint.

**Figure 48: Goals Used by Human Tutors.**

### 6.2.3 Requirement for and Sufficiency of Preemption Based on Context.

I claim that *two* basic mechanisms are sufficient to produce tutorial discourse: default passage through a network of states and exception rules to generate preemptive jumps between these states. I shall refer to the application of these two mechanisms as preemptive planning. There are several ways of testing the sufficiency of these two mechanisms. The primary test proposes that if the range of tutorial text is dependent upon the two basic planning mechanisms, the quality of text produced is a measure of the sufficiency of the claim. Obviously then, the success of this design can be tested



only by using the system with large numbers of students and in a variety of domains. Though I have not performed extensive tests with the system, I have illustrated that natural, human quality tutorial discourse can be produced by a default network of tutoring states within which a series of preemption jumps are allowed.

The requirement that two mechanisms are sufficient is substantiated by looking at the contrapositive: one of the two components is not needed by the system to plan tutoring text. I have described a dual control mechanism: default execution and preemption; therefore, I need to ask if both mechanisms are necessary.

First I will test the requirement of the meta-rules. Recall that default execution moves through the tutoring states, its path determined by an augmented transition network (ATN). The action of the meta-rules creates an change of state that results in a new direction for the discourse and possibly new topics to explore. For example, the strategic meta-rule S3-DESCRIBE is enabled when there is evidence that the student does not know the topics well enough to continue a question-and-answer routine or that the teaching strategy must be abridged to present material from the knowledge base (see Section 4.3). This rule is triggered when the threshold of permitted incorrect student answers has been reached or when the student seems confused. Without this strategic meta-rule, the system would continue to question the student and proceed through a list of preestablished topics. The rule enables the system to quit an existing and presumably futile approach and to begin a descriptive passage about a topic from the knowledge base.

Meta-rules provide a flexible way to shift the expository style, yet they are not the only available computational method to perform this shift. For instance, one might restrict the usage of specific styles to an explicit number of iterations, or meter the use of each through a recursive stack of tutoring approaches (see next section). However, the discourse produced through this method would not be as responsive to student difficulties and would not take into consideration current activities or the state of the discourse. That is, the choice of whether or not to use a particular style must be derived from the student and discourse context. In summary, meta-rules are necessary to preemptive planning because their loss would weaken the effectiveness and flexibility of the system.

In a similar way, default execution through tutoring states can be shown to be necessary for tutoring text generation, i.e., no discourse would be coherent or consistent if limited to a single style, or if generated by undefined and random traversal through the rhetorical and pedagogical decisions embodied by the tutoring states. That is, some form of default transition through some set of decisions is required.

I claim that any system other than Meno-tutor that produces tutoring behavior will have mechanisms that (1) achieve some default behavior, and (2) permit preemption from this default path. I suggest that other and more sophisticated mechanisms will be developed in other tutoring systems, yet they will have the same functionality as the ones I have defined or will themselves be computationally equivalent to the ones I have

defined.

#### **6.2.4 Requirements for Feedback.**

Machine tutoring requires feedback; it is an interaction based on the comparison of actual performance with some "desired" performance, nominally the expert's performance. In such an interaction, there is always the potential for unexpected disturbances (e.g., a student who is too ill-prepared to even discuss the topic). But the tutor that is sensitive to a variety of student errors and processes and has predictive power based on working with many nontraditional students, can use nuances in student feedback to adequately predict student knowledge. For instance, a correct answer generally implies understanding of the subject matter. However, a correct answer that follows a series of incorrect answers, or a correct answer which has an incorrect component, may imply that, though the student knows the some aspect of the subject, he lacks thorough knowledge of it.

Meno-tutor performs elementary processes to "compute" its response to the student based on feedback. It is able to reveal the student's missing or confused information; however, it is not yet able to reduce that difference in an efficient way. Recognizing the nature of the gap between expert and student knowledge and effectively reducing it require constant vigilance and readjustment of the system's view of the student's knowledge. Such extensive feedback has not yet been effectively realized, yet it seems a possible extension to the system.

#### **6.2.5 Absence of a Requirement for a Recursive Stack.**

Stevens and Collins [1977], Grosz [1980], and McKeown [1983] suggest that textual structure is recursive in nature. Stevens and Collins make a convincing argument for the necessity of a recursive stack to simulate the behavior of human tutors in structuring their teaching goals. They suggest that a goal is added (pushed on) to the stack and pursued by the tutor until a new goal is triggered. The new goal is triggered if it is implicit in a topic mentioned by the student or if it preempts the current goal (e.g., supply missing knowledge before supplying incorrect knowledge). When the second goal is triggered into the space of the discourse, it, too, is added to the stack and becomes the current goal. The second goal is worked on until it is satisfied. Then it—is removed (popped) from the stack. Only then is the original goal worked on until it is satisfied.

McKeown has proposed that rhetorical structures can be used recursively to generate a variety of different hierarchically structured texts from a limited number of discourse plans. She uses recursion to describe the rhetorical predicates available to a speaker and suggests that several paragraph-level schemas can account for rhetorical structure over a large range of expository texts; she expresses these schemas as context-free grammars, the primitives of which are predicates such as "attributive," "amplification," and "restriction."

Tutoring discourse – especially because of its multiperson aspect – requires more varied and flexible structures than predicates, applied recursively, can provide. In fact, tutoring style and topics are generated in reaction to the student's knowledge and his input to the system. To ask how one tutors is akin to asking how an inspector catches a bank robber; the actions of the law enforcer are so dependent upon the activities of the bank robber as to suggest that the law enforcer has no agenda of his own except to react to the activities of the bank robber [Collins, et al., 1975].

It can be demonstrated that human tutors do return to previously mentioned topics and goals [Collins, et al., 1975]. However, they do so in a rather unpredictable manner and not in a manner that implies strict recursive protocol. In tutoring, a topic presented once is often discussed again; sometimes it turns up by way of some intermediate topic. The original topic is not set aside, rather it is "retired" while a related topic is discussed. The original subject is returned to, especially to show the relation between it and the intermediate topic. However, if explicit clarification is not needed (i.e., if the student's correct response implies that he might have figured out the original topic), then the original topic will not be mentioned. Thus, a stack mechanism that enforces repetition of the earlier topic is too rigid for tutoring discourse.

### 6.3 Features of the System

Meno-tutor system offers advantages over alternative approaches to text planning for explanation and tutoring. These advantages follow from rejection of the "retrieval-orientated" approach to discourse generation in favor of a "context-dependent" model of discourse planning. These terms and the features of the system will be described below.

#### *Rejection of Retrieval-Orientated Discourse Production*

The first contribution of Meno-tutor is to recognize the importance of using student input to plan the discourse. Most machine teaching systems are designed around a prescribed set of concepts or tutoring approaches. The design of such systems anticipates student's answers and prescribes a response based on explicit rules. I call this type of system "retrieval-orientated." Multisentential text generation and explanation systems also are designed in this way: they generate text for the stereotypical user (e.g., Mann & Moore [1981], McKeown [1980], and Swartout [1977]). These systems are intended primarily as single initiative systems; they answer questions from within a limited set, or describe data from within an existing knowledge base. These systems are not meant to be responsive to unanticipated input from the student. In a discourse system, particularly a tutoring system, lack of responsiveness can become a serious problem.

For instance, if a system describes the play-by-play activities in a football game and a listening student is unaware of the goals of the game, then such a student might make incoherent comments or questions about the game. Unless the system is alerted to the fact that the student needs to understand the goals of the game before learning its plays, the system might take no meaning from the student's comments. In such a case, an inconsistency develops between what the student knows and the system's model of his knowledge. The student's input, which is the only data the system has about his knowledge, will provide wrong and misleading clues.

#### *Context-Dependent Tutoring*

The "context-dependent" model of tutoring, on the other hand, is designed to recognize the student's knowledge. It defines tutoring as a process based on a tight relation between discourse planning and the model of the user's knowledge. In Meno-tutor the system is aware of topics that might be too "obvious" or too difficult for the student and has information about what topics might be misleading or trivial for him. Control of the discourse is flexibly maintained by using preemption planning based on the models of the student, the discourse, and the subject area. In Meno-tutor, for instance, inconsistencies between the student's answers and the evolving model of his knowledge, are used to refocus the tutor's attention and to redirect the discourse.

Misconceptions in the user's knowledge are important to the success of explanation or text generation systems, too. A misconception in the user's understanding of the domain might prohibit him from understanding a system's response or cause the system to produce an inappropriate response. For instance, the TEXT system [McKeown, 1980] generates paragraph-length explanations in response to a user's request for information about a "ships and submarines" knowledge base. However, the system is not designed to recognize misconceptions. In one case, a user asked about the draft of a submarine and the system responded with a paragraph-long description about draft and displacement of ships in general, and submarines in particular, noting that submarines do not have draft or displacement. Had the system been alerted to the fact that the user either did not know that the ship in question was a submarine, or that submarines don't have draft and displacement, it might have avoided producing a paragraph of irrelevant and useless data. In addition, if the system had explicitly stated the user's misconception and had clarified his notions of draft and displacement, it would have improved the user's knowledge of the subject area, and have improved his interactions with the system.

#### *Planning Based on Partial Knowledge*

Meno-tutor's third contribution is the ability to represent and reason from partial knowledge about the student's knowledge. For instance, the system might recognize that the model of student knowledge is not complete (i.e., the register \*competence\_confirmed is not set) and yet not yet have a complete plan of what to do next. The system can tolerate this kind of ambiguity and can wait until several iterations to resolve it. The ability to have unresolved and partial information about the student is critical in tutoring. By its nature, tutoring is based on partial and evidential knowledge and is

characterized by varying degrees of ignorance on the part of the tutor as to what the student understands and what he does understand. Other tutoring systems do not properly capture this aspect of reasoning from uncertainty. They often enforce preplanned or rigid response behavior.

Rather than plan from "complete" knowledge, Meno-tutor proposed a coordinated set of states that are flexible in the sense that they provide, but because of the meta-rules, do not enforce a fixed schema for planning the response. The states may be entered from a variety of previous states and do not themselves provide complete discourse planning. When information about the student's knowledge is only partially available, as in the case where a misconception is triggered, the planned response, to verify the misconception, can be postponed or preempted until more information is obtained. For example, if a misconception is triggered, the tutor can do a variety of things. It can 1) ignore the triggered misconception, 2) question the student directly about it, 3) ask questions about related topics to further establish the kind of misconception, or 4) generalize the misconception to other cases or domains where the student's logical inconsistency will become evident. There is no clear path to take; the discourse decision to explore or repair the misconception can be postponed indefinitely or handled immediately. If the system's information incompletely defines the next state, it can be supplemented or reinforced by sampling the environment, the history of the discourse, or model of the student. In this way, a given planning approach will not force a response of the system: the approach can be preempted until additional information more clearly defines the appropriate approach.

#### *Text Planning without Backtracking*

This research has demonstrated that text planning is possible without backward reasoning or backtracking. That is, Meno-tutor did not explicitly plan a set of responses that were later abandoned. Rather, it maintained as current a number of implicit plans for the discourse direction and used its simple preemptive planning mechanism to eliminate all but one approach. The addition of backtracking might enable more sophisticated reasoning about the user's knowledge and the pedagogical goals, but for the two domains explored in this research, the limited, forward reasoning described by this research seemed sufficient.

It is conceivable that dialogues in other domains might need a more complex control structure or will require more complex planning mechanisms than those provided by Meno-tutor. This can be determined only by extensive testing of the system in a variety of domains.

#### *A Tool for Cognitive Science Research*

Meno-tutor offers an opportunity to develop and refine the rules of tutoring discourse. It provides a tool and a computational framework within which pedagogic and rhetorical conventions of tutoring might evolve and be elaborated upon. The states and preemptive rules provided in this study are preliminary. It has been my goal to provide

a mechanism that other researchers can use and modify as new domains and different students are tested. It is quite likely that further researchers will add specialized and customized constructs that will improve discourse performance in specific subject areas. This will both clarify and elaborate on the initial tool set provided here.

#### **6.4 Limitations of the System**

One of the most obvious shortcomings of this work is that it was not applied to any large task. The theory proposed here needs to be applied to a complex domain where its range of flexibility and relevance can be tested. The success of such a study would depend on the ability of the designer to construct the data base needed to capture the domain knowledge and on his ability to develop methods for proper access to relevant topics and rules within that knowledge. The initial investment required to write the annotations and update the tutoring rules to a new domain will be repaid in terms of more accessible and relevant tutoring discourse.

Other shortcomings in Meno-tutor have little or no theoretical merit. They call for extensions to the system and enhancements to the mechanisms that I expect will improve its functioning. These are described below, along with specific changes and improvements to be made.

- Natural language processing was not implemented. The language generation is canned and the student input is simulated. Interfacing the Meno-tutor to the surface language generator is expected to be a straightforward, if not trivial task. It will require defining terms specific to tutoring and the subject under discussion, and expressing these in the language of the dictionary of the surface language generator. These definitions have been begun as a part of the pseudo-dictionary in Meno-tutor. Building a natural language understanding facility for the system would not be trivial; understanding the terms used by a student to explain his knowledge is a complex task. A limited and perhaps restricted vocabulary could be built and initially tested.
- The student model was tested for a limited set of topics. Extension to the student model, to enable it to handle a variety of topics, will require additional mechanisms to ensure that increasing the complex relations between topics is computationally handled.
- The post-processor slots of the meta-rules were never used. These slots would hold functions to update the discourse or student model after a meta-rule was run. Recording this data was handled by functions in the post-processing slots of the states. It would be instructive to see how much more refined and tractable the discourse and student models could be if the slots of the meta-rules were used, instead.

### **6.5 Implications of the Research**

This research supplies solutions to some problems of discourse generation. It explores the problems of reasoning and learning about the user's knowledge and develops a context-dependent system that goes beyond merely giving advice. Chapter 5 illustrates the results of computational methods presented here and suggests ways to work with a student who is, for instance, knowledgeable, or unknowledgable, or confused.

The implications of these results go beyond tutoring. They apply to explanation systems, question-answering systems, on-line help, and consultant systems. The research indicates the need for an intelligence interface to define the user's level of understanding before supplying him with answers. It also points out the need for a discourse system to converse with a user at his own level of understanding. These results suggest that any expert system engaging a user in a discourse must have an active agent to keep track of the user's knowledge, to measure his competence, and to consider which style and topics are appropriate for him. Thus, any intelligent interface must have the ability of a skilled tutor: it must reason about the user and work with his difficulties and confusions.

### **6.6 Implications of the Research to Education**

Implications of this research for education are profound; they extend into every phase of education as we know it today and then into a few new ones. The potential here is that effective teaching systems, with human-like tutoring abilities, might one day become available, not only in classrooms in the near future, but in libraries, supermarkets, museums and anywhere that people congregate. Encyclopedic sources of information are already becoming routinely available through micro-computers and a tutoring system, such as proposed in this dissertation, might provide the key to individualizing these systems to the student's own level and style of learning.

In the classroom, intelligent I.C.A.I. tutors will probably coexist with teachers; by coexist, I mean both might teach the subject, but the best use of the machine will be to supplement normal school-room lectures and reading assignments. Machines could provide the initial exposure to subjects, but their most effective use would be as sensitive tutors, familiar with a variety of misconceptions and able to explore a range of nontraditional learning processes with a student. See Sections 2.4 and 2.5 for a discussion of how to represent expert and student knowledge. The machine can extend the teacher's introductory material by offering instructive tools, e.g., an encyclopedic reference about related topics, a graphic display of a mathematical formula, or a simulation of a chemical process. The student can choose how and when to use the system and can learn from it at his own pace. The systems might also be used "in reverse" by student teachers who would use the machines to simulate student behavior

and thereby gain experience in evaluating student answer and diagnosing errors.

Any learning experience will be most productive if the student is actively involved in restructuring and reorganizing his own knowledge during the learning experience [Piaget, 1971]. In terms of a computer tutor, this means that the student should be in control of retrieving or modifying the instructional tools made available by the machine, should visualize his own errors and should be active in their repair. This notion of active learning or "learning by doing" is formally expressed as the constructivist educational paradigm and is based on the premise that *knowing* is an activity of the student and *knowledge* is a construction. According to Piaget, knowledge is neither an item which the student acquires, neither is it something about the object which the student must grasp. Rather, knowledge is seen as something constructed by the student; it is an active, dynamic and relational activity, not a static item. When this philosophy is applied to the design of a computer tutor, it suggests that the system must respond actively to a student who constructs and transform his own knowledge.

In addition to engaging the student in an active learning process, the high quality computer tutor must use the student errors as a source and motivating force behind the student's learning. Without errors, knowledge itself would remain static. According to the constructivist paradigm, errors are the vehicle which enables the student to move from his current concept to a more advanced, more equilibrated level in which stable or balanced knowledge is achieved. Learning takes place through repeated attempts by the student to compensate for the disturbances (disequilibria) which occur when his inadequate assimilation schemes (templates by which knowledge is incorporated) are applied. Errors provide a way and a motivation for the student to examine and rework his knowledge. A teaching system that incorporates this philosophy would allow the student to make errors, yet would actively participate in their correction and repair. One of the goals of such a process is to allow for and to foster student autonomy.

Such a philosophy requires what I call "error-motivated" teaching strategies based on understanding the cognitive processes involved in learning the subject area. See Section 2.4 for a discussion about how to represent a student's knowledge in a teaching system. The philosophy suggests a system which actively involves the student in transforming objects in his environment and one which encourages a student to change, rearrange, or enhance his own knowledge while interacting with the system. Such an interaction places the system in the position of a dutiful servant - presenting the appropriate simulation, strategy, or tool, at the appropriate time. In such a system, the judge of its pedagogical success will be the system's ability to understand the student and to appraise its own effectiveness. Its merit will not focus on the sophistication of its tools, nor the availability of technological "bells and whistles" (e.g., windows, mice or high-resolution graphic screens).

On the other hand, current systems, by and large, are based on "errorless learning" which is a derivative of the behavioral educational paradigm. These systems place a premium on success and correct responses - some authors boast that students can't make a mistake within their system. These systems are not designed to allow



students to see their errors as potentially fruitful, leading to a new direction, or as giving a sudden new insight and the behavioral curriculum is organized into simple, discrete increments which are intended to eliminate the opportunity for and recovery from error. Another consequence of errorless learning is that the student fails to learn what to ask on his own or to develop or test his own hypotheses. The goal of such a system is to correct and reward the student while working in an environment where reward and punishment are seen as the activities that shape human behavior. The machine implications of this view are that the system's principle task is to present knowledge of others, e.g., to compare the student's information with that of an expert in the field, the teacher, or the author of the system. In the constructivist viewpoint, the intent of the system is to define and reveal the student's knowledge, and then to elaborate upon it.

To build a fully integrated computer tutor based on a constructivist paradigm will require the collaborative efforts of both educator and computer scientist. It will require integrating the theoretical foundations of education with the computational foundations of computer science. It will also require an understanding of the psychological and theoretical bases of teaching and learning. Such a collaboration can provide powerful tutoring systems available in virtually every domain: any domain in which an expert system can be built is a potential domain source for an instructional system. See Section 1.1 and Section 2.4, particularly the discussion of GUIDON for an example of a teaching system built from an expert system. Intelligent teaching systems are now being developed, e.g., in algebra, geometry, physics, electronics and medicine. Some systems are based on advanced teaching strategies, others reflect rather primitive mechanisms. See Chapter 2, particularly Sections 2.5 and 2.6 for a discussion of existing systems. Today's system, by and large, are based on error-less learning; once the more sophisticated error-motivated systems begin to emerge, they will more successfully join the existing educational environment.

For instance, intelligent computer tutors might be linked to central computers, made available in instructional technology laboratories through the community, and made available around-the-clock - something like perpetual instructors. The computers might be linked through local area networks that provide access to file servers and peripherals such as printers and plotters. The education provided from these networked systems would include:

1. Computer-based instruction (e.g., teaching problem solving in physics via the computer; instruction in foreign languages).
2. Computer-based research (e.g., studies of human-computer interactions).
3. Instruction by simulation (e.g., economic and engineering simulations, modeling chemical experiments, simulating mental processes, etc.).

4. Applications programming (e.g., music composition, speech input and output, input and manipulation of visual images).
5. Real-time applications (e.g., using computers to control experimental apparatus and to collect experimental data).
6. Teacher training (e.g., evaluation of instructional programs, development of courseware, experiments with tutoring programs, diagnosing student errors by observing his interactions with student modeler programs).
7. On-line communication (e.g., peer teaching through electronic mail, on-line tutors and consultants).
8. Data-base libraries (e.g., construction access and manipulation of data for storage in generally available libraries).

The potential breadth of instructional software is great; the potential quality of the software to be produced is equally great. Once both potentials are met, the one for quality and the one for quantity of instructional materials, the resulting systems will change the very nature of education.

Could education itself become obsolete? It is likely that "good" teaching will not be replaced. However, it is very likely that "bad" teaching will be replaced and that education as we know it will change. But if we are to develop high-quality instructional systems to work in harmony with good teachers, we ought to clarify our definition of "good" teaching, which is, or ought to be, our model for the high-quality instructional system. For instance, teaching today is tracked and modulated by the clock, is constrained to large student teacher ratios, and is restricted to a classroom. These aspects of teaching obfuscate its essential nature; the classroom is not an essential part of teaching, nor is the constraint of time nor classroom size. The system proposed in this thesis, and others like it will not be restricted by time, classroom or numbers of students. They will make teaching available any place, any time, and from the "best" of teachers; that is, software packages prepared by leaders in each field will be made routinely available to any person. In order to develop excellent teaching systems, I have begun the process of isolating aspects of teaching that are essential to the realization of teaching and those that are not. In Section 1.3 I began to define tutoring as a phenomenon apart from its connection to the classroom; these same characterizations might be applied to teaching.

More effective and more obvious collaboration between the computer scientists and the educator will begin the process of developing more pervasive, more compelling and more intelligent teaching systems. Such a successful collaboration will allow us to temper the rigor and austerity of current systems and to sensitize them toward more flexible teaching interactions. Included in such design is the need to modulate of the use of technological "trappings" and electronic "bells and whistles" in instructional systems; we need to find what makes computer education more effective, not what makes it more

entertaining; what makes computer education more profound, not what makes it more addictive. Computer technology alone does not hold the key to revolutionizing education; computer technology provides an opportunity for the skillful blending of educational philosophy and computation to produce systems that might revolutionize the technology of education.

### **6.7 Further Research Directions**

The research begun here might be continued along several directions. One direction might include building up the tutor's knowledge of a single subject. For example, if the target domain is the looping constructs of Pascal, we would extend the system so that the tutor could explore a full range of errors, plans, and misconceptions that a novice programmer might present. The goal of such a system would be to have the tutor react dynamically to nuances of the student's input and to recognize uncertainties in his knowledge. Primary in such a research effort would be to build the system so that it could distinguish between a variety of potential student misconceptions.

A second direction for developing this research might be to further develop the variety of responses possible. For instance, we might want to enhance the system's ability to produce explanationa and analogies, and to retrieve and modify examples appropriately for a particular student. Customizing examples has been done in the several domains (see, for instance, Rissland & Soloway [1981]).

A third approach would be to extend the system so that it can monitors a user's interactions with an expert system. In this application of the system, the discourse planning facility would make inferences about the student's level of knowledge and would direct his interactions with the expert system based on those inferences. For instance, the system might make a judgement about the user's ability to use the system commands correctly and would adjust both the style of its discourse and its execution of those commands based on its interpretation of the user's ability.

Each extension, briefly outlined above, requires elaboration and extensions of the mechanisms of this reserach. For instance, in the first extension, new meta-rules would be defined to handle making decisions between alternative misconceptions. In the second extension, the expert model would be revised to reflect the optimal way for the user to use commands to interact with the system.

## CHAPTER VII

## BIBLIOGRAPHY

- Atkinson, R. C., & Wilson, H. A. (Eds.), *Computer-Assisted Instruction*, Academic Press, New York, 1969.
- Allen J., *A Plan-Based Approach to Speech Act Recognition*, Ph. D. Thesis, University of Toronto, 1979, available as Technical Report 121/79, Department of Computer Science, University of Toronto, Toronto, Canada, 1979.
- , "Recognizing Intentions from Natural Language Utterances," in M. Brady & R. Berwick (Eds.), *Computational Models of Discourse*, the M.I.T. Press, Cambridge, Mass., 1983.
- Arbib, M. A., "Parallelism, Slides, Schemas, and Frames," in W. E. Hartnett (Ed.), *Systems: Approaches, Theories, Applications*, D. Reidel Publishing Company, 1977.
- Austin, H., *A Computational View of Physical Skill*, Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, Mass., 1976.
- Barr, A., Beard, M., & Atkinson, R. C., "The Computer as a Tutorial Laboratory: The Stanford Bip Project," in the *International Journal Of Man-Machine Studies*, 8, 1976.
- Bates, M., Brown, G., & Collins, A., "Socratic Teaching of Causal Knowledge and Reasoning," Bolt Beranek and Newman, Cambridge, Mass., 1978.
- Bitzer, D., Braunfeld, P., & Lichtenberger, W., "PLATO: An Automatic Teaching Device," *IRE Transactions on Education*, E4, 157-161, Dec, 1961.
- Bonar, J., *Understanding the Bugs of Novice Programming*, Ph.D. Dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst, Mass., 1984.
- Bonar, J., Ehrlich, K., Soloway, E., & Rubin, E., "Collecting and Analysing On-line Protocols from Novice Programmers," *Behavioral Research Methods and Instrumentation*, (14), 203-209, 1982.
- Bower, G. H., Black, J., & Turner, T., "Scripts in Text Comprehension and Memory," in *Cognitive Psychology*, 1, 177-220, 1979.
- Bobrow, D. C., Kaplan, R. M., & Kay, M., "GUS, a Frame-driven Dialog System," in *Artificial Intelligence*, 8, (2), 155-173, 1977.
- Brachman, R., "A Structural Paradigm for Representing Knowledge," Report 3605, Bolt

- Beranek and Newman, Cambridge, Mass., 1978.
- Brachman, R., Ciccarelli, E., Greenfield, N., & Yonke, M., KLONE Reference Manual, Report 3848, Bolt Beranek and Newman, Cambridge, Mass., 1978.
- Brown, G., "A System to Process Dialogue: A Progress Report," Laboratory for Computer Science, Technical Report .79, M.I.T., Cambridge, Mass., 1976.
- Brown, J. S., & Burton R., "Multiple Representation of Knowledge for Tutorial Reasoning," in D. C. Bobrow & A. Collins (Eds.), *Representation and Understanding: Studies in Cognitive Sciences*, Academic Press, New York, 1975.
- Brown, J. S., & Bell, A., "SOPHIE: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting (An Example of A.I. in C.A.I.)," *International Journal of Man-Machine Studies*, (7), 1977.
- Brown, J. S., & Burton, B., "Diagnostic Models for Procedural Bugs in Basic Mathematical Skills," *Cognitive Sciences*, 2, pp. 155-192, 1978.
- Brown, J. S., Burton, B., Housmann, C., Goldstein, I., Huggins, B., & Miller, M., "Aspects of a Theory for Automated Student Modelling," Bolt Beranek and Newman, Technical Report 3549, Cambridge, Mass., 1977.
- Brown, J. S., Rubinstein, R., & Burton, R., "Reactive Learning Environments for Computer Assisted Electronics Instruction," Technical Report 3314, Bolt Beranek and Newman, Cambridge, Mass., 1976.
- Brown, J. S., & Lehman, K., "Repair Theory: A Generative Theory of Bugs in Procedural Skills," in *Cognitive Science*, Vol. 4, No. 4, 1980.
- Bruce, B. C., "Generation as a Social Action," in R. Schank & B. Nash-Webber (Eds.), Theoretical Issues in Natural Language Processing, pp. 74-77, 1975.
- Burton, R., & Brown, J. S., "An Investigation of Computer Coaching for Informal Learning Activities," *International Journal of Man-Machine Studies*, 11, 1978, also in Sleeman & Brown (Eds.) *Intelligent Tutoring Systems*, Academic Press, Cambridge, Mass., 1982.
- Carbonell, Jaime R., *Mixed-Initiative Man-computer Instructional Dialogues*, Technical Report 1971, Bolt Beranek and Newman, Cambridge, Mass., June 1970.
- Carr, B., & Goldstein, I., "Overlays: a Theory of Modelling For Computer Aided Instruction," M.I.T., A.I., Memo 406, Feb., 1977.
- Chase, W., & Simon, H., "The Mind's Eye in Chess," In W. G. Chase (Ed.) *Visual Information Processing*, Academic Press, New York, 1973.

- \_\_\_\_\_, "Perception in Chess," *Cognitive Psychology*, 4, pp. 55-81, 1981.
- Clancey, W., *Transfer of Rule-Based Expertise Through Tutorial Dialogue*, Ph.D. Dissertation, Department of Computer Science, Stanford University, Stanford, CA, 1979a.
- \_\_\_\_\_, "Dialogue Management for Rule-Based Tutorials," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1979b.
- \_\_\_\_\_, "Tutoring Rules for Guiding a Case Method Dialogue," *International Journal of Man-Machine Studies*, 11, 1979, also in Sleeman & Brown (Eds.), *Intelligent Tutoring Systems*, Academic Press, Cambridge, Mass., 1982.
- Clancey, W., & Letsinger, R., "Neomycin: Reconfiguring a Rule-Based Expert System for Application to Teaching," Report STAN-CS-82-908, 1982, also appeared in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1981.
- Cohen, P., *On Knowing What to Say: Planning Speech Acts*, Ph.D thesis, available as Technical Report 118, University of Toronto, Toronto, Canada, 1978.
- Collins, A., & Stevens, A., "Goals and Strategies of Effective Teachers," Technical Report, Bolt Beranek and Newman, Cambridge, Mass., 1980.
- Collins, A., Warnock, E., & Passafiume, J., "Analysis and Synthesis of Tutorial Dialogues," Report 2789, Bolt Beranek and Newman, Cambridge, Mass., 1974, also in *Psychology of Learning and Motivation*, Vol. 9, Academic Press, 1975.
- Conklin, J., *Data-Driven indelible Planning of Discourse Generation Using Saliency*, Ph.D. Dissertation, Department of Computer and Information Sciences, University of Massachusetts, Amherst, Mass., 1983.
- Flores, C., & Winograd, T., *Understanding Computers and Cognition*, to appear.
- Forbus, K., & Stevens, A., "Using Qualitative Simulation to Generate Explanations," Report 4490, Bolt Beranek and Newman, Cambridge, Mass., 1981.
- Furth, H. G., *Piaget and Knowledge*, Prentice Hall, Englewood Cliffs, New Jersey, 1969.
- Genesereth, M., *Automated Consultation for Complex Computer Systems*, Ph.D. Dissertation, Department of Computer Science, Harvard University, 1978.
- \_\_\_\_\_, "The Role of Plans in Intelligent Teaching Systems," *International Journal of Man-Machine Studies*, 11, 1979, also in Sleeman & Brown (Eds.), *Intelligent Tutoring Systems*, Academic Press, Cambridge, Mass., 1982.
- Gerhart, S., "Knowledge about Programs: A Model and Case Study," in *Proceedings of*

- the Conference on Reliable Software*, SIGPLAN Notes, Vol. 10, No. 6, 1975.
- Getner, D., & Norman, D., "The Flow Tutor: Schemes for Tutors," Technical Report 7702, Center for Human Information Processing, University of San Diego, San Diego, Calif., 1977.
- Goldstein, I. P., "The Genetic Graph: A Representation for the Evolution of Procedural Knowledge," in *International Journal of Man-Machine Studies*, 1, 11, pp. 51-78, 1979.
- , "The Computer as Coach: An Athletic Paradigm for Intellectual Education," AI Memo 389, Massachusetts Institute of Technology, Cambridge, Mass., 1977.
- Grice, H. P., "Logic and Conversation," in P. Cole & J. L. Morgan (Eds.), *Syntax and Semantics: Speech Acts*, Vol. 3, Academic Press, New York, 1975.
- Grosz, B. J., "Focusing and Description in Natural Language Dialogues," in A. Joshi, et al. (Eds.), *Elements of Discourse Understanding: Proceedings of a Workshop on Computational Aspects of Linguistic Structure and Discourse Setting*, Cambridge University Press, Cambridge, 1980.
- , "Discourse Analysis" in D. Walker (Ed.), *Speech Understanding Research, Final Technical Report*, Stanford Research Institute, 1976.
- Haugeland, J., "Semantic Engines: An Introduction to Mind Design," in J. Haugeland (Ed.), *Mind Design*, The M.I.T. Press, Cambridge, Mass, 1982.
- Lakatos, I., *Proofs and Refutations*, Cambridge University Press, London, 1976.
- Larkin, J., McDermot, J., Simon, D. P., & Simon, H., "Expert and Novice Performance in Solving Physics Problems," *Science*, Vol. 208, 20, 1980.
- Lehnert, W. G., "A Conceptual Theory of Question Answering," in the *Fifth Conference of the International Joint Committee on Artificial Intelligence*, 1977.
- Mark, W., "Mapping Requests into Explanation Forms," Consul Note 6, USC/Information Sciences Institute, 1981.
- Mann, W., & Moore, J., "Computer Generation of Multiparagraph English Text," *American Journal of Computational Linguistics*, Vol. 7, 1, Jan-March, 1981.
- McDermott, D., "Artificial Intelligence Meets Natural Stupidity," in J. Haugeland (Ed.), *Mind Design*, The MIT Press, Cambridge, Mass., 1982.
- McDonald, D., *Natural Language Production as a Process of Decision Making Under Constraint*, Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Mass., 1980.

- \_\_\_\_\_, "Natural Language Generation as a Computational Problem: an Introduction," in M. Brady & R. Berwick (Eds.), *Computational Models of Discourse*, MIT Press, Cambridge, Mass, 1983.
- \_\_\_\_\_, "Description Directed Control: Its Implications for Natural Language Generation," in *Computational Linguistics*, in the series *Computation and Mathematics with Applications*, Vol 9, No 1, pp 111-129, Pergamon Press, Ltd., Great Britain, 1983.
- McDonald, D., & Conklin, J., "Salience as a Simplifying Metaphor for Natural Language Generation," in *Proceedings of the Conference of the American Association for Artificial Intelligence*, Pittsburg, Penn., 1982.
- McKeown, K., "Generating Relevant Explanations: Natural Language Responses to Questions about Data Base Structure," in *Proceedings of the Conference of the American Association for Artificial Intelligence*, Stanford University, 1980.
- \_\_\_\_\_, "Recursion in Text and its Use in Language Generation," in *Proceedings of the Conference of the American Association for Artificial Intelligence*, Pittsburg, Penn., 1982.
- Miller, M., "A Structured Planning Environment," *International Journal of Man-Machine Studies*, 11, 1979, also in Sleeman & Brown (Eds.), *Intelligent Tutoring Systems*, Academic Press, Cambridge, Mass, 1982.
- Miller, M., & Goldstein, I., "Structured Planning and Debugging," in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, M.I.T., Cambridge, Mass., 1977.
- Minsky, M. (Ed.), *Semantic Information Processing*, The MIT Press, Massachusetts Institute of Technology, Cambridge, 1968.
- \_\_\_\_\_, "Why People Think Machines Don't," in *Technology Review*, Massachusetts Institute of Technology, Cambridge, Mass., November/December 198
- Mitre Corporation, "Computers in the Instructional Process: Report of an International School", Extend Publications, Ann Arbor, Michigan, 1972.
- Papert, S., "Teaching Children to be Mathematicians Versus Teaching about Mathematics," Artificial Intelligence Laboratory, A.I. Memo 249, MIT, Cambridge, Mass., 1971.
- Peelle, H., & Riseman, E., "The Four Faces of HAL: A Framework for Using Artificial Intelligence Techniques in Computer-Assisted Instruction," in Institute of Electronic and Electrical Engineers (IEEE), *Transactions on Computers*, 1975.
- Perrault, R., & Cohen, P., "Planning Speech Acts, Technical Report," Department of



- Computer Science, University of Toronto, 1977.
- Piaget, J., *Genetic Epistemology*, W. W. Norton, Press, 1971.
- Plato, *Laches, Protagoras, Meno, and Euthydemus*, W. R. M. Lamb, translator, Harvard University Press, Cambridge, Mass., 1922.
- Pollack, M.E., & Hirschberg, "User Participation in the Reasoning Processes of Expert Systems" in *Proceedings of the Conference of the American Association for Artificial Intelligence*, Pittsburgh, Penn., 1982.
- Pope, C., "Education of Emotionally Disturbed Children: A Critique of Behaviorist Trends and an Examination of a Constructivist Alternative," Ed.D. Dissertation, School of Education, University of Massachusetts, Amherst, Mass., 1982.
- Pople, H. E., "The Synthesis of Composite Hypotheses in Diagnostic Problem Solving: An Exercise in Hypothetical Reasoning," in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 1977.
- Rich, C., & Shrobe, H., "Initial report on a LISP Programmer's Apprentice," *IEEE Transactions on Software Engineering*, 4, 6, pp. 342-376, November, 1978.
- Rich, E., *Building and Explaining User Models*, Ph.D. Dissertation, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Penn., 1979.
- Rissland (Michner) E., *Epistemology, Representation, Understanding and Interactive Exploration of Mathematical Theories*, Ph.D. Dissertation, Department of Mathematics, M.I.T., Cambridge, Mass., 1977.
- Rissland, E., "Structure of Mathematical Knowledge," Technical Report 472, MIT Artificial Intelligence Lab, Cambridge, Mass., 1980.
- \_\_\_\_\_, "Understanding Understanding Mathematics," *Cognitive Science*, Vol. 2, 4, 1978.
- Rissland, E. & Soloway, E., "Constrained Example Generation: A testbed for Studying Learning", in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 1981.
- Rubin, E., "The Bug Finder," unpublished master's project, Computer and Information Sciences, University of Massachusetts, Amherst, Mass., 1981.
- Sacerdoti, E., "The Non-linear Nature of Plans," in *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, Tbilis, Georgia, USSR, 1975.
- Shortliffe, E. H., *MYCIN: A Rule-based Computer Program for Advising Physicians regarding Antimicrobial Therapy Selection*, Ph.D Dissertation, Medical Information

- and Sciences, Stanford University, Stanford, Calif., 1974.
- , *Computer-based Medical Consultations: MYCIN* Elsevier Press, New York, 1976.
- Skinner, B. F., *Beyond Freedom and Dignity*, Knopf, New York, 1971.
- Sleeman, D. H., & Smith, M., "Modelling Student's Problem Solving," in *Artificial Intelligence Journal*, 33, pp. 22-27, 1981.
- Stansfield, J., Carr, B. & Goldstein, I., "Wumpus Advisor I: A First Implementation of a Program that Tutors Logical and Probabilistic Reasoning Skill," M.I.T. A.I. Lab Memo, No. 381, 1976.
- Soloway, E., Ehrlich, K., & Black J., "Beyond Numbers: Don't Ask 'How Many'... Ask 'Why'," in CHI '83: Human Factors in Computing Systems, ACM, 1983.
- Soloway, E., Bonar, J., & Ehrlich, K., "Cognitive Factors in Programming: an Empirical Study of Looping Constructs," Tech Report 81-10, Department of Computer Science, University of Massachusetts, Amherst, Mass., 1981a.
- Soloway, E., Bonar, J., Woolf, B., Barth, P., Rubin, E., & Erlich, K., "Cognition and Programming: Why your Students Write Those Crazy Programs," *Proceedings of the National Educational Computing Conference*, NECC, No. Denton, Tx, 1981b.
- Soloway, E., Woolf, B., Barth, P., & Rubin, E., "MENO-II: An Intelligent Tutoring System for Novice Programmers," in *The Seventh International Joint Conference on Artificial Intelligence*, Vancouver, Canada, 1981c.
- Soloway, E., & Woolf, B., "Problems, Plans, and Programs," in *Proceedings of the Eleventh ACM Technical Symposium on Computer Science Education*, 1979.
- Soloway, E., *Learning Interpretation Generalization: A Case Study in Knowledge Directed Learning*, Ph.D. Dissertation, Computer and Information Sciences, University of Massachusetts, Amherst, Mass., 1978, also available as Technical Report 78-13, 1978.
- Stevens, A., & Collins, A., "The Goal Structure of a Socratic Tutor," Report 3518, Bolt Beranek and Newman, Cambridge, Mass., 1977, also appears in *Proceedings of the Association for Computing Machinery Annual Conference*, 1977.
- Stevens, A., Collins, A., & Goldin, S., "Diagnosing Student's Misconceptions in Causal Models," Technical Report 3786, Bolt Beranek and Newman, Cambridge, Mass., 1978, also in *International Journal of Man-Machines Studies*, 11, 1978 and in Sleeman & Brown (Eds.), *Intelligent Tutoring Systems*, Academic Press, Cambridge, Mass, 1982.
- Sussman, G., "A Computational Model of Skill Acquisition," Technical Report 297,

- Artificial Intelligence Laboratory, M.I.T., Cambridge, Mass., 1973.
- Swartout, W., "A Digitalis Therapy Advisor with Explanations," in *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 1977.
- , "The GIST Behavior Explainer," in *Proceedings of the Third Conference of the American Association for Artificial Intelligence*, Washington, D.C., 1983.
- Wilensky, R., "Meta-Plans," *Proceedings of the First Conference of American Association of Artificial Intelligence*, Palo Alto, California, August, 1981.
- Winston, P. H., "Learning Structural Descriptions from Examples," Project MAC, Technical Report MAC-TR-76, M.I.T., Cambridge, Mass., 1970.
- Woods, W. A., "Transition Network Grammars for Natural Language Analysis," *Communications of the Association for Computing Machinery*, Vol. 13, No. 10, October, 1970.
- Woolf, B., & McDonald, D., "Human-Computer Discourse in the Design of a PASCAL Tutor", *CHI '83: Human Factors in Computer Systems*, ACM, New York, 1983.

## A P P E N D I X A

## TRACES OF THE OPERATIONS OF THE MENO-TUTOR

The following traces of the operations of the Lisp program, Meno-tutor were made in the production of three of the discourses cited in this dissertation. In each discourse, Meno-tutor enters a state indicated by a preceding dollar sign and left directed arrow. The tutor evaluates the functions in the slots of each state and then moves to the next state, indicated by the state below. The length of the arrow indicates the level of the tutoring state, i.e., Pedagogic States have long arrows, Strategic States medium length ones, and Tactical States, the smallest arrows. A meta-rule is indicated between two upward (^) and two downward pointing (vv) arrows. The action of the meta rule moves the tutor to the state directly below the meta-rule name. The student's input, typed in capital letters, was simulated during discourse production and the text following the semicolons was added after the trace to explain the operations of the system.

; The first trace was produced by Meno-tutor while generating  
 ; the discourse in Figure 1, Chapter 1. This discourse was described  
 ; in great detail in Section 4 and will not be elaborated upon here.

```
? (run)
  | <===== $introduce
  | <===== $introduce_domain
    | <===== $realize_specific_intro

      MENO-TUTOR: Do you know what the climate is like
                  Washington and Oregon?

                        *****
  | <===== $realize_evaluate_input

      STUDENT: IS IT COLD?
      ^
      T2
      vv
  | <===== $realize_explicit_incorrect_acknowledgement

      MENO-TUTOR: No

                        *****
  | <===== $teach_knowledge
    | <===== $realize_teach_specific_knowledge
```

**MENO-TUTOR:** It's rather mild there

```

*****
| <=====          $teach_knowledge
  ^
  S1-EXPLORE
  vv
| <=====          $explore_competency
  | <=====          $realize_exploratory_question

```

**MENO-TUTOR:** Can you guess about the rainfall there?

```

*****
| <=====          $realize_evaluate_input

```

**STUDENT:** NORMAL, I GUESS.

```

  ^
  T2-A.INCORRECT
  vv
| <=====          $realize_explicitly_incorrect_acknowledgement
  ^

```

T6-A.IMPLICITLY

```

  vv
| <=====          $realize_explicit_incorrect_acknowledgement

```

**MENO-TUTOR:** Well, ...

```

*****
| <=====          $teach_knowledge
  ^
  S3-DESCRIBE
  vv
| <=====          $describe_domain
  | <=====          $realize_describe_general_knowledge

```

**MENO-TUTOR:** The Japan Current which starts in southeastern Pacific goes along the coast of Japan and across the North Pacific ending up off the coast of Washington and Oregon.

```

*****
| <=====          $realize_question_dependency

```

**MENO-TUTOR:** How do you think that affects the climate in Washington and Oregon?

\*\*\*\*\*

| <===== \$realize\_evaluate\_input

*STUDENT:* IT'S PROBABLY RAINY.

^^

T1-A.CORRECT

v^v

**\*\*END\*\*CONTROLLER\*\***

? (doff)

"dribble-off"

; The second trace was made while generating the rainfall discourse  
 ; in Figure 40, Section 5.2.1. The high level  
 ; transitions are discussed along with the register which  
 ; enabled their transitions.

? (run)

```
| <===== $introduce
| <===== $introduce_domain
  | <===== $realize_specific_intro
```

**MENO-TUTOR:** Do you think it rains much in Oregon?

```
*****
| <===== $realize_evaluate_input
```

**STUDENT:** NO.

^^

**T5-JETTISON**

vv

; This rule, namely \*response\_is\_incorrect, is fired  
 ; because two registers, \*misconception\_is\_triggered  
 ; and \*response\_is\_incorrect are set.

```
| <===== $teach_knowledge
  ^^
```

**S6-JETTISON**

vv

; This rule fires for the same reason  
 ; as the previous one.

```
| <===== $tutor
  ^^
```

**P2-HACK**

vv

; This rule is fired by the  
 ; single predicate \*misconception\_is\_triggered

```
| <===== $hack_misconception
| <===== $propose_misconception
  | <===== $realize_propose_misconception
```

**MENO-TUTOR:** Why do you think it doesn't rain much  
 in Oregon?

\*\*\*\*\*

**STUDENT:** Because the neighboring states

are dry.

| <===== \$propose\_misconception  
 ^

**S4-VERIFY**

vv

; This rule is triggered by the register \*suspicion\_confirmed  
 ; because the student has indicated a reason  
 ; for his ill-founded belief.

| <===== \$verify\_misconception  
 | <===== \$realize\_verify\_misconception

; The state \$realize\_verify\_misconception is entered but does produce an  
 ; utterance because there is no misconception in the expert knowledge base  
 ; associated with the student's mistaken reasoning. The slot Domain Model  
 ; in the state \$realize\_verify\_misconception lists the topics connected  
 ; to the current topic and does not mention "neighboring states."

| <===== \$verify\_misconception  
 ^

**S5-REPAIR**

vv

; This rule is triggered by the register  
 ; \*misconception\_verified, which is set by  
 ; the state \$realize\_verify\_misconception

| <===== \$repair\_misconception  
 | <===== \$realize\_repair\_misconception

**MENO-TUTOR:** That is a misconception.

\*\*\*\*\*  
 | <===== \$realize\_grain\_of\_truth\_correction

**MENO-TUTOR:** The neighboring states are dry  
 but, you can not generalize from that  
 It does in fact rain a lot in Oregon.

\*\*\*\*\*  
 | <===== \$teach\_knowledge  
 ^

**S1-EXPLORE**

vv

; This rule is triggered because the previous  
 ; topic, "rain in Oregon", has been completed



; through the previous misconception probe  
 ; and other related topics remain to be discussed  
 ; in the expert knowledge base.

| <===== \$explore\_competency  
 | <===== \$realize\_exploratory\_question

*MENO-TUTOR*: Can you guess what causes the rain there?

\*\*\*\*\*

| <===== \$realize\_evaluate\_input

*STUDENT*: Well, there are high mountains in  
 the region and the states border  
 the Pacific Ocean.

^^

T5-JETTISON

v^v

**\*\*END\*\*CONTROLLER\*\***

? (doff)  
 "dribble-off"

; This third trace was produced by Meno-tutor while generating the discourse  
 ; in Figure 42 in Section 5.2.2.

? (run)

```
| ===== $introduce
| <===== $introduce_domain
| <===== $realize_specific_intro
```

*MENO-TUTOR:* Do you know that GRADES in line 9 is  
 a control variable for the WHILE loop

```
*****
| <===== $realize_evaluate_input
```

*STUDENT:* YES

^^

T1-A.CORRECT

vv

```
| <===== $realize_explicit_correct_acknowledgement
```

*MENO-TUTOR:* That's right

```
*****
| <===== $teach_knowledge
| ^
```

S1-EXPLORE

vv

```
| <===== $explore_competency
| <===== $realize_exploratory_question
| ^
```

T10-R.EXPLORE

vv

```
| <===== $realize_role_exploratory_question
```

*MENO-TUTOR:* Does that mean that GRADES  
 in Line 9.  
 a) initiated before the loop?  
 b) added into a sum within the loop?  
 c) modified within the loop?  
 d) both a and c?

```
*****
| <===== $realize_evaluate_input
```

*STUDENT:* C

```

      ^
      T1-A.CORRECT
      v
| <===== $realize_explicit_correct_acknowledgement

      MENO-TUTOR: That's right

      *****

| <===== $teach_knowledge
      ^
      S6-JETTISON
      v
| <===== $tutor
      ^
      P3-COMPLETE
      v
| <===== $complete
| <===== $complete_topic
  | <===== $realize_complete_topic

      MENO-TUTOR: You seem to understand about the
                  control variable for the WHILE loop

      *****

| <===== $complete_topic
**END**CONTROLLER**
?(doff)
"dribble-off"

```

## A P P E N D I X B

## LISP PRETTY PRINTING OF THE TUTORING STATES

## THE TUTORING STATES

```
<to pretty print the states from the Meno-tutor
type
  (route-out 'states (describe_states)) >
```

\*\*\*\*\*

## THE PEDAGOGICAL STATES

\*\*\*\*\*

## \$introduce

```
  update:          (*time_spent *success_of method)
  next_states:     ($tutor $hack_misconception $complete)
```

## \$tutor

```
  update:          (*time_spent *success_of method)
  next_states:     ($hack_misconception $complete)
```

## \$hack\_misconception

```
  update:          (*time_spent *success_of method)
  next_states:     ($tutor $complete)
```

## \$complete

```
  update:          (*time_spent *success_of method)
  next_states:     ($introduce)
```

\*\*\*\*\*

## THE STRATEGICAL STATES

\*\*\*\*\*

## \$introduce\_domain

```
  constraints:     (*questions_on_one_topic expository_mix)
```

|                                     |   |
|-------------------------------------|---|
| discourse_model:                    | (number_of_repetitions number_of_corrections)   |
| student_model:                      | (general_knowledge Michelin_rating)             |
| next_states:                        | (nil)   |
| <b>\$introduce_continuing_topic</b> |   |
| constraints:                        | (*questions_on_one_topic expository_mix)        |
| discourse_model:                    | (number_of_repetitions number_of_corrections)   |
| student_model:                      | (general_knowledge Michelin_rating)             |
| next_states:                        | (nil)   |
| <b>\$explore_competency</b>         |   |
| constraints:                        | (*questions_on_one_topic expository_mix)        |
| discourse_model:                    | (number_of_repetitions number_of_corrections)   |
| student_model:                      | (general_knowledge Michelin_rating)             |
| next_states:                        | strategy_for_tutoring                           |
| <b>\$teach_knowledge</b>            |   |
| constraints:                        | (*questions_on_one_topic expository_mix)        |
| discourse_model:                    | (number_of_repetitions number_of_corrections)   |
| student_model:                      | (general_knowledge Michelin_rating)             |
| next_states:                        | (nil)   |
| <b>\$describe_domain</b>            |   |
| constraints:                        | (*questions_on_one_topic expository_mix)        |
| discourse_model:                    | (number_of_repetitions number_of_corrections)   |
| student_model:                      | (general_knowledge Michelin_rating)             |
| next_states:                        | strategy_for_tutoring                           |
| <b>\$propose_misconception</b>      |   |
| constraints:                        | (*questions_on_one_topic expository_mix)        |
| discourse_model:                    | (number_of_repetitions number_of_corrections)   |
| student_model:                      | (general_knowledge Michelin_rating)             |
| next_states:                        | (\$verify_misconception)                        |
| <b>\$verify_misconception</b>       |   |
| constraints:                        | (*questions_on_one_topic expository_mix)        |
| discourse_model:                    | (number_of_repetitions number_of_corrections)   |
| student_model:                      | (general_knowledge Michelin_rating)             |
| next_states:                        | (\$verify_misconception \$repair_misconception) |
| <b>\$repair_misconception</b>       |   |
| constraints:                        | (*questions_on_one_topic expository_mix)        |
| discourse_model:                    | (number_of_repetitions number_of_corrections)   |
| student_model:                      | (general_knowledge Michelin_rating)             |
| next_states:                        | (nil)   |

```
$complete_topic
  constraints:      (*questions_on_one_topic expository_mix)
  discourse_model: (number_of_repetitions number_of_corrections)
  student_model:  (general_knowledge Michelin_rating)
  next_states:    (nil)
```

\*\*\*\*\*

## THE TACTICAL STATES

\*\*\*\*\*

```

$realize_specific_intro
  domain_model:      (complexity connectivity expected_competency)
  student_model:     (explicit_errors implicit_misconceptions)
  utter_elmts:       (##introduce_domain *major_topic)
  next_states:       ($realize_evaluate_input)
  post_processing:   (record_student_response update_discourse_model)

$realize_general_intro
  domain_model:      (complexity connectivity expected_competency)
  student_model:     (explicit_errors implicit_misconceptions)
  utter_elmts:       (##question_model *current_topic)
  next_states:       ($realize_evaluate_input)
  post_processing:   (record_student_response update_discourse_model)

$realize_exploratory_question
  domain_model:      (complexity connectivity expected_competency)
  student_model:     (explicit_errors implicit_misconceptions)
  utter_elmts:       (##question_model *current_topic)
  next_states:       ($realize_evaluate_input)
  post_processing:   (record_student_response update_discourse_model)

$realize_role_exploratory_question
  domain_model:      (complexity connectivity expected_competency)
  student_model:     (explicit_errors implicit_misconceptions)
  utter_elmts:       (##question_role *major_topic *role_current_topic)
  next_states:       ($realize_evaluate_input)
  post_processing:   (record_student_response update_discourse_model)

$realize_teach_specific_knowledge
  domain_model:      (complexity connectivity expected_competency)
  student_model:     (explicit_errors implicit_misconceptions)
  utter_elmts:       (##specific_teach *current_role *ibid_role)
  next_states:       (nil)
  post_processing:   (complete_question update_student_model)

$realize_teach_general_knowledge
  domain_model:      (complexity connectivity expected_competency)
  student_model:     (explicit_errors implicit_misconceptions)
  utter_elmts:       (##unknown)
  next_states:       (nil)

```

```

    post_processing:                (complete_question update_student_model)

$realize_teach_dependent_knowledge
  domain_model:                   (complexity connectivity expected_competency)
  student_model:                   (explicit_errors implicit_misconceptions)
  utter_elmts:                     (##unknown)
  next_states:                     (nil)
  post_processing:                 (complete_question update_student_model)

$realize_question_dependency
  domain_model:                   (complexity connectivity expected_competency)
  student_model:                   (explicit_errors implicit_misconceptions)
  utter_elmts:                     (##question_factors *major_topic)
  next_states:                     ($realize_evaluate_input)
  post_processing:                 (record_student_response update_discourse_model)

$realize_explicit_correct_acknowledgement
  domain_model:                   (complexity connectivity expected_competency)
  student_model:                   (explicit_errors implicit_misconceptions)
  utter_elmts:                     (##positive)
  next_states:                     (nil)
  post_processing:                 (complete_question update_student_model)

$realize_implicit_correct_acknowledgement
  domain_model:                   (complexity connectivity expected_competency)
  student_model:                   (explicit_errors implicit_misconceptions)
  utter_elmts:                     (##positive_weakly)
  next_states:                     (nil)
  post_processing:                 (complete_question update_student_model)

$realize_emphatic_correct_acknowledgement
  domain_model:                   (complexity connectivity expected_competency)
  student_model:                   (explicit_errors implicit_misconceptions)
  utter_elmts:                     (##positive_strongly ##role_major_topic ##brief_conju
  next_states:                     (nil)
  post_processing:                 (complete_question update_student_model)

$realize_explicit_incorrect_acknowledgement
  domain_model:                   (complexity connectivity expected_competency)
  student_model:                   (explicit_errors implicit_misconceptions)
  utter_elmts:                     (##negative)
  next_states:                     (nil)
  post_processing:                 (update_student_model)

$realize_implicit_incorrect_acknowledgement
  domain_model:                   (complexity connectivity expected_competency)

```



```

student_model: (explicit_errors implicit_misconceptions)
utter_elmts:   (##negative_weakly)
next_states:   (nil)
post_processing: (complete_question update_student_model)

$realize_evaluate_input
domain_model: (complexity connectivity expected_competency)
student_model: (explicit_errors implicit_misconceptions)
utter_elmts:   nil
next_states:   (nil)
post_processing: (update_teaching_done)

$realize_grain_of_truth_correction
domain_model: (complexity connectivity expected_competency)
student_model: (explicit_errors implicit_misconceptions)
utter_elmts:   (##repeat_user ##negation ##role_conflict *misc1 *ro
next_states:   ($realize_suggest_new_concept)
post_processing: (complete_question update_student_model)

$realize_is_correct_and_elaborate
domain_model: (complexity connectivity expected_competency)
student_model: (explicit_errors implicit_misconceptions)
utter_elmts:   (##positive ##conjunction)
next_states:   ($realize_describe_specific_knowledge)
post_processing: (complete_question update_student_model)

$realize_describe_general_knowledge
domain_model: (complexity connectivity expected_competency)
student_model: (explicit_errors implicit_misconceptions)
utter_elmts:   (##describe_role *current_topic)
next_states:   ($realize_question_dependency)
post_processing: (update_description_made)

$realize_describe_specific_knowledge
domain_model: (complexity connectivity expected_competency)
student_model: (explicit_errors implicit_misconceptions)
utter_elmts:   (##specific_describe *role_second_topic)
next_states:   ($realize_question_dependency)
post_processing: (update_description_made)

$realize_describe_dependent_knowledge
domain_model: (complexity connectivity expected_competency)
student_model: (explicit_errors implicit_misconceptions)
utter_elmts:   (##specific_describe *current_topic)
next_states:   (nil)
post_processing: (update_description_made)

```

```

$realize_question_reflexive_factor
  domain_model: (complexity connectivity expected_competency)
  student_model: (explicit_errors implicit_misconceptions)
  utter_elmts:   (##question_reflexive_factor *feature *second_topic)
  next_states:  (nil)
  post_processing: (record_student_response update_discourse_model)

$realize_propose_analogy
  domain_model: (complexity connectivity expected_competency)
  student_model: (explicit_errors implicit_misconceptions)
  utter_elmts:   (##unknown)
  next_states:  (nil)
  post_processing: (update_analogy_made)

$realize_suggest_example
  domain_model: (complexity connectivity expected_competency)
  student_model: (explicit_errors implicit_misconceptions)
  utter_elmts:   (##unknown)
  next_states:  (nil)
  post_processing: (update_example_made)

$realize_propose_misconception
  domain_model: (complexity connectivity expected_competency)
  student_model: (explicit_errors implicit_misconceptions)
  utter_elmts:   (##question_function *misc1 *function_misc1 *major_to)
  next_states:  (nil)
  post_processing: (record_student_response test_prop_misc)

$realize_verify_misconception
  domain_model: (complexity connectivity expected_competency)
  student_model: (explicit_errors implicit_misconceptions)
  utter_elmts:   (##implication_conjunction ##question_role *misc2)
  next_states:  (nil)
  post_processing: (record_student_response test_ver_misc)

$realize_repair_misconception
  domain_model: (complexity connectivity expected_competency)
  student_model: (explicit_errors implicit_misconceptions)
  utter_elmts:   (##declare_misc)
  next_states:  ($realize_grain_of_truth_correction)
  post_processing: (record_repair_misc)

$realize_suggest_new_concept
  domain_model: (complexity connectivity expected_competency)
  student_model: (explicit_errors implicit_misconceptions)
  utter_elmts:   (##suggest_concept *locate_concept *new_concept *mis

```

|                                 |   |
|---------------------------------|---|
| next_states:                    | (nil)   |
| post_processing:                | (nil)   |
| <b>\$realize_complete_topic</b> |   |
| domain_model:                   | (complexity connectivity expected_competency) |
| student_model:                  | (explicit_errors implicit_misconceptions)     |
| utter_elmts:                    | (##confirm_competence *function_of_topic)     |
| next_states:                    | (nil)   |
| post_processing:                | (update_discourse_model)                      |

## A P P E N D I X C

## HIGH LEVEL CODE OF THE MENO-TUTOR

```

#####
;#                                     #
;#      MENO SYSTEM                       #      MENO.INI
;#                                     #
;#                                     #####
;#      File to initialize MENO:
;#          A program to
;#      generate tutorial discourse
;#
;#          Beverly Park Woolf
;#          Spring, 1983
;#                                     #
#####
;
;      File name: MENO.ini
;
;      This file loads in the entire system.
;      File are loaded in the following order:
;
;      CLISP FILES
;      MACRO.EVL          >> - macros to make hacking fun
;      LIBRARY.LSP        >> - good lisp functions
;      READFNS.EVL        >> - useful read functions
;      PRINTFNS.EVL       >> - useful print routines
;
;      TEMP-DATA
;      RAIN2.GRR           >> - KL-ONE network of topics for understanding rainfall
;      DATA.EVL          >> - KL-ONE network of topics for understanding rainfa
;
;      TUTOR FILES
;      LINGUA.EVL          >> - functions to communicate between MENO,MUMBLE a
;      RAIN.DIC            >> - simulated MUMBLE dictionary for rainfall
;      PASCAL.DIC          >> - simulated MUMBLE dictionary for PASCAL
;      BLANK.DIC           >> - simulated MUMBLE dictionary for erasing previous di
;      MODELEV            >> - functions to model the student and the discourse
;      STUDENT.EVL         >> - functions to handle interaction between Meno and stu
;      DEFSTATES.EVL       >> - functions to define Tutoring States
;      STATES.LSP          >> - storage of State structures
;
;      CONTROL FILES
;      CNTRL.EVL           >> - the controller functions;

```

```

; DEFRULES.EVL      >> - functions to define transition rules
; RULES.LSP        >> - Transition rules
; PATH.EVL         >> - an obscure and nearly extinct file XXX
;
; WEATHER FILES
; MAINFNS          >> - EMPTY!!
; HELPERS          >> - auxiliary database access files
; MYKLONE          >> - tunes up access routines to most important roles/attribu
;
; KLONE FILES
; KLONE:INIT.LSP   >> - initializes the creation of concepts
; KLONE:CREATE.LSP >> - creates KLONE nodes and edges
; KLONE:RETRIEVE.LSP >> - retrieves KLONE nodes and edges
; KLONE:DEF.LSP    >> - defines KLONE networks, both interactively
; KLONE:DEFHELP.LSP >> - auxiliary funtions to define KLONE network
; KLONE:GRR.LSP    >> - graph processing language based on array an
; KLONE:PPNET.LSP  >> - pretty print the KLONE network
;
;#####;
;          Functions to load the whole system
;
;#####;
(setq load_the_whole_system nil
  files_loaded nil)
;
;
(defun meno_eval (file)

  (print file)
  (eval-file file)
  ]
;
(defun load_the_states ()
  (let (load_whole_world t)
    (terpri) (princ "CLISP and TUTOR files: ") (terpri)
    (fc-nil (star meno_eval)
      `(macro defstates lingua rain.dic model student readfns
printfns data)
    )
; (load-file 'states.lsp) storage of tutoring states
  ]
;
(defun load_the_control ()
  (terpri) (princ "CONTROL files: ") (terpri)
  (fc-nil (star meno_eval)
    `(cntrl defrules path)

```

```

    )
; (load-file 'rules.lsp)
]
;
(defun load_pascal ()
  (comment routine to load the states and dictionary
    for tutoring in Pascal)
  (terpri)(princ "PASCAL DOMAIN files:")(terpri)
  (fc-nil (star meno_eval)
    '(blank.dic pasfile pascal.dic )
  ]

(defun load_dial40 ()
  (comment routine to load the rules and dictionary
    for Dial40)
  (terpri)(princ "Editing the meta-rules:")(terpri)
  (fc-nil (star meno_eval)
    '(blank.dic dial40.dic )
  ]
;
; routines to save functions as system is refined
;
(ndefun save (atom)
  (insert (quote patch21) (list lat) )
  )
;
;
(ndefun ssave (atom)
  (replace (quote patch21) (list lat) )
  )
;
;
;#####;
;
;      Start the system
;
;#####;
;
;  Initializes control and structures
;  within a fresh lisp
;
;
;
(load_the_states)
;
;

```







```

;
;
;
(defun initialize_topics_list ()
  (comment (sets *topics_list* to the list of individuated objects in the
             kl-one space which is currently loaded))
  (print (setq $topics_list (sort-objects-by-michelin
                          (get-all-subconcepts (quote **individuated_concepts) sp))
  ;
;
;
(initialize_topics_list)
;
;
(defun initialize_salutation ()
  (terpri)
  (princ "*****      Welcome to MENO      *****")
  (terpri)
  (terpri)
  (princ "          Don't forget to look at          ")
  (terpri)
  (princ "          patches.lsp          ")

  (terpri)
  (princ "          before hacking today          ")
  (terpri)
  (princ " Happy Hacking !!!          ")
  (terpri)(terpri)
  (princ " Todays date: ")(princ (date))
  (terpri)(terpri) ]
;
;
;
(initialize_salutation)
(bells)
(bells)

```

```

;#####;
;
;      MENO SYSTEM      #      CNTRL.EVL
;
;      #
;      #####
;
;      File to establish global
;      control over MENO's responses
;
;      Beverly Woolf
;      March 1, 1983
;
;
;#####;
; Top Level Bindings for the Controller
;#####;
;
(defun initialize_discourse ()
;
;#####;
;
;      Initialize Registers
;
;#####;
(setq *wrong_answer_threshold nil
      $utterance nil
      *new_topics nil
      *topics_complete nil
      *competence_confirmed nil
      *threshold_wongs 2
      *salient_topic 'y
      *terminate_strategy nil
      *suspicion_confirmed nil
      *misconception_triggered nil
      *misconception_verified nil
      *question_complete t
      *student_in_doubt nil
      *completion_acknowledged t
      response_is_correct nil
      response_is_incorrect nil)
;
; several HACK global variables
(setq
;
;
;      should result from scan of Knowledge Base
;
;

```

```

    *additional_roles_exist t
    *related_topics      t
    *answer_available    t
    *dependency_exists   t
    *inference_exists    t
    *salient_topic       t
;
;      others
;
;      *correct_after_wrong_answer nil
)
;#####
;
;      Counters
;#####
(setq      *ply_count          0
    *utterance_count    0
    *num_wrong_answers  0
)
;
;
;      (putprop *counters nil `#topics_questioned)
;      (putprop *counters 0 `#wrong_answers)
;      (putprop *counters 0 `#right_answers)
;      (putprop *counters nil `#topics_described)
;      (putprop *counters nil `#topics_examined)
;
;
;      (putprop *misconception_triggered nil `#misc_evidence)
;      (putprop *misconception_triggered nil `#misc_confirmed)
;#####
;
;      Flags
;#####
(setq
    root_state      '$introduce
    *trace-states   t
]
(initialize_discourse)
(setq *end_dialogue 20)           ; number of plys allowed in discourse
;

```



```

(defun the_controller (state)
  (comment (repeatedly plans and replans utterances, ultimately
            sending a finished product to mumble. Called by MENO_TUTOR))
  (let (*ply_count 0)
    (repeat ()
      (process_node (eval state))
      (setq state (car next_states)
              level (get_structure_type state)
              *ply_count (add1 *ply_count))
    )
    until (eq *ply_count *end_dialogue))
  )
  (terpri)
  **END**CONTROLLER**)

;
;
;
(defun process_node (state)
  (comment (interpret action at leach phase of the planner)
            (generate a trace of the system)
            (called by the_controller))
  (setq current_state state)
  (let (level (get_structure_type state))
    (case level
      ($ped_state
        (let (*left-margin 5)
          (if *trace-states (printc "| <=====") "(describe_state
            (process_node1 state ^$ped_state)))
        ($strat_state
          (let (*left-margin 10)
            (if *trace-states (printc "| <=====") "(describe_state state))
            (process_node1 state ^$strat_state)))
        ($stact_state
          (let (*left-margin 15)
            (if *trace-states (printc "| <=====") "(describe_state state))
            (process_node1 state ^$stact_state)))
      (mbug "process_node has found an unknown type"
        state)
    )
    ; case
  ]

;
;
;
;
;

```

```

;#####;
;
;           The Workers
;
;#####;
;
(defun process_node1 (state level)
  (comment (executes the state and state descriptors
            moves to lower states which make further refinements
            on the speech template. At the lowest level text is
            uttered )
           (control is returned to parent state by accessing
            parent from a plist attributed to parent)
           (called by dispatch_on_constarints))
  ;
  (let (qstate (get_state_name state))
    (cond ((preempt_path qstate level) nil)
          ;
          ;           execute state if preempt is not called
          ;
          (t (case level
              ((quote $$ped_state) (execute_state state))
              ((quote $$strat_state) (execute_state state))
              ((quote $$tact_state)
               (if (context_is_appropriate state)
                   (realize_utterance state)
                   (post_process_utterance (get_slot_access state 'post_processing) qstate)
                   (setq next_states (get_next_states state))
               )
              (format " inappropriate state " state)))
          ) ; cond
  ;
  ; if next state was not declared by a preemption rule or default state
  ; then return to parental state
  ;
  (if (or (null next_states) (equal next_states (quote (nil))))
      (pop_planning_level qstate level))
  ]
  ;
  ;
  (defun execute_state (state)
    (setq next_states (plist (get_state_name state)))
    )
  ;
  (defun pop_planning_level (qstate level)

```

```

(comment (finds the next highest planning state which performs
         similar planning activities)
         (auxilliary funtions are in path.evl)
         (called by process_node1 when next_states is not declared))
(setq next_states
  (list (search_parental_units qstate
        (find_state_descriptor qstate level)
        ]
        ;
(defun context_is_appropriate ()
  (comment checks the slots domain_model
           and student_model to see if the context is appropriate
           for this state)
  (and (test_domain_model (get_slot_access state 'domain_model) qstate)
        (test_student_model (get_slot_access state 'student_model) qstate)
        ]
        ;
;
(defun test_domain_model (lst_functions qstate)
  (comment (evaluates the functions in the slot
           domain_model of the state)
           (typical functions are *complexity* of expert
           knowledge *connectivity* *expected_competency*)
           )
  (fc-and (star (lambda (domain_function)
                 (format "          " domain_function)
                 (apply domain_function (list qstate))))
           lst_functions
           ]
           ;
;
(defun test_student_model (lst_functions qstate)
  (comment (evaluates the functions in the slot
           student_model of the state)
           (typical functions are *explicit_errors*
           *implicit_misconceptions*)
           )
  (fc-and (star (lambda (student_function)
                 (format "          " student_function)
                 (apply student_function (list qstate))))
           lst_functions
           ]
           ;
;
(defun post_process_utterance (lst_functions qstate)
  (comment evaluates the function in the post_process slot of the
           tactical state. Functions include update the discourse model

```

```
        or update student record)
;
(fc-nil (star (lambda (post_process_function)
              (format "                " post_process_function)
              (apply post_process_function (list qstate))))
        lst_functions)
]
```



```

;#####;
;
;      MENO SYSTEM      #      PATHEVL      #
;
;      #
;#####
;
;      File to plan the preempt
;      conditions in MENO's response
;
;
;      Beverly Woolf      #
;      April 18, 1983
;
;
;      #
;      #
;      #
;
;      Department of Computer and Information Sciences
;      Graduate Research Center
;      University of Massachusetts
;      Amherst, MA 01003
;
;#####;
;
; Lists the states which will be considered for preemption
;
; (setq list_preemptable_states
;   `($tutor $complete $explore_competency $teach_knowledge
;     $realize_exploratory_question
;     $realize_evaluate_input
;     $realize_explicit_correct_acknowledgement
;     $realize_explicit_incorrect_acknowledgement))
;
;
; (defun preempt_path (qstate level)
;   (comment (check present state against a sampling of the environment to see if cont
;     If preemption is called for (/, the) state is established and discourse/stude
;     the_controller is resumed) (state refers to state name not state structure)
;   (if (memq qstate list_preemptable_states)
;     (let* (enabled_rules
;           (check_preconditions (check_prior_state qstate level))
;           triggered_one
;           (condcar enabled_rules))
;       (if triggered_one (act_on_transition triggered_one) (reset_discourse_model trig
;         triggered_one))))
;
;#####;
;

```

```

;           Functions to make the decisions
;
;#####;
;
(defun check_preconditions (rules_list)
  (comment (returns a list of transition rules in which the preconditions
            are satisfied))
    (fc-condcons (star test_preconditions)
                 rules_list)
  )
;
(defun check_prior_state (qstate level)
  (comment (returns a list of rules which apply by virtue
            of *state* being the *prior state* of that rule)
            ( only rules at the same level as *state* are checked))
  )
;
(let (rules_list (get_rules_of level))
  (fc-condcons
   (star (lambda (rule)
           (if (element? qstate (get_slot_access (eval rule) 'prior_state))
               rule )
          )
        rules_list)
  )
;
(defun test_preconditions (rule)
  (comment (tests precondition of rule)
            (expects a conjunction of variable which will
            be tested for a positive value)
            (called by preempt_path through check_preconditions)
  )
  (if (fc-and (star eval)
              (get_slot_access (eval rule) 'preconditions)
  )
      rule ]
;
;
;
(defun get_states_of (level)
  (case level
    (' $$ped_state  $$pedagogical_states)
    (' $$strat_state $$strategical_states)
    (' $$tact_state  $$tactical_states)
    (format " wrong level indicated by get_states_of " level)
  )
)

```

```

;
(defun get_rules_of (level)
  (case level
    ($$strat_state $$strat_rules)
    ($$tact_state $$tact_rules )
    ($$ped_state $$ped_rules)
    (format "Wrong level indicated in *get_rules_of " level)
  )
)
;
(defun act_on_transition (rule_run)
  (comment (acts on rule chosen to shift discourse style and
            or topics)
           (called by preempt_path and effectively
            shifts the state (or topics) and resets dialogue
            before returning control to the_controller)
  )
  (if rule_run
    (terpri)
    (writeln "  ^^ ")
    (writeln "          " rule_run)
    (writeln "  vv ")
    (act_on_rule rule_run)
  )
)
;
;
;
;
(defun act_on_rule (rule)
  (eval (get_slot_access (eval rule) 'action))
)
;
;
;#####;
; Functions to move planning to further refinement
; or further abstraction
;
;#####;
;
;
(defun find_state_descriptor (qstate level)
  (comment (returns a descriptor of the type of
            pedagogical activity performed by qstate)
           (called by pop_planning_level and later functions
            called by update_discourse_model))
  (test_for_state_subset qstate (find_list_of_states level))
)
;
;

```

```
(defun test_for_state_subset (qstate state_subsets)
  (comment tests the *subset of states* for the one which contains
    *qstate*)
  ;
  (let*
    (subset
      (fc-condcons
        (star (lambda (qstate lat)
              (if (element? qstate (eval lat)) lat)))
        (star qstate)
        state_subsets)
      )
    (make_atom subset)
    ) ;let
  ]
  ;
```



```

        (putprop '$utterance curr-utterance '#curr-utterance)
        (putprop '$utterance $topics_list '#topics_list)
    )
]
;
(defun build_utterance ()
  (comment (builds an utterance by adjoining the tutoreme-
            rhetorical phrase meant for tutoring – with a member of
            a realization class in order to create a r-spec for MUMBLE
            called by realize_utterance)
    )
  (insert_in_utterance (get_elements))
)
;
;
;
;
;
(defun insert_in_utterance (elmts)
  ;###Hack this is what would be sent to MUMBLE
  ;
  (putprop '$utterance elmts '#curr_utterance)
  ]
)
(defun get_elements ()
  (comment (picks out the realization class elements from inside
            the tactical states. These elements are passed by way of
            insert_in_utterance into the property list of $utterance)
            (called by build_utterance))
  (if (type_tact_state current_state)
      (get_slot_access current_state 'utter_elmts))
  ]
)
(defun type_tact_state (state)
  (comment ( check that *state* is a $$taactical planner) )
  (eq (get_structure_type state) '$$tact_state)
  ]
)
;
;
;
(defun enough_said? ()
  (comment (true if third stage descriptor has no more
            elements for that utterance.
            called by realize_utterance.
            could eventually make a decision based on
            michelin ratings))
    t
  ]
)
; ###hack find who moves elements
; from tact stage to #current_utterance

```



```
(defun get_dict_entry (entry)
  (let (val (eval entry))
    (cond
      ((listp val)(get_element_from_class val))
      (t val )
    )
  )
;
(defun get_element_from_class (r_class)
  (stack.pop r_class)
;##Hack we really want a circular que
]
```