A HYPERMEDIA-BASED INTELLIGENT COMPUTER-ASSISTED INSTRUCTION MODEL

by

Ogazi Rose Joshua

A thesis
presented to the University of Manitoba
in partial fulfilment of the
requirements for the degree of
Doctor of Philosophy
in
Computer Science

Winnipeg, Manitoba, CANADA, 1996

ISBN 0-612-13227-7

Canada

Name _____

Dissertation Abstracts International and *Masters Abstracts International* are arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation or thesis. Enter the corresponding four-digit code in the spaces provided.

COMPUTER SCIENCE
SUBJECT TERM

0984
SUBJECT CODE

UMI

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS
Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION
General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523
Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS
Language
- General 0679
- Ancient 0289
- Linguistics 0290
- Modern 0291

Literature
- General 0401
- Classical 0294
- Comparative 0295
- Medieval 0297
- Modern 0298
- African 0316
- American 0591
- Asian 0305
- Canadian (English) 0352
- Canadian (French) 0355
- English 0593
- Germanic 0311
- Latin American 0312
- Middle Eastern 0315
- Romance 0313
- Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY
Philosophy 0422
Religion
- General 0318
- Biblical Studies 0321
- Clergy 0319
- History of 0320
- Philosophy of 0322

Theology 0469

SOCIAL SCIENCES
American Studies 0323
Anthropology
- Archaeology 0324
- Cultural 0326
- Physical 0327

Business Administration
- General 0310
- Accounting 0272
- Banking 0770
- Management 0454
- Marketing 0338

Canadian Studies 0385
Economics
- General 0501
- Agricultural 0503
- Commerce-Business 0505
- Finance 0508
- History 0509
- Labor 0510
- Theory 0511

Folklore 0358
Geography 0366
Gerontology 0351
History
- General 0578
- Ancient 0579
- Medieval 0581
- Modern 0582
- Black 0328
- African 0331
- Asia, Australia and Oceania 0332
- Canadian 0334
- European 0335
- Latin American 0336
- Middle Eastern 0333
- United States 0337

History of Science 0585
Law 0398
Political Science
- General 0615
- International Law and Relations 0616
- Public Administration 0617

Recreation 0814
Social Work 0452
Sociology
- General 0626
- Criminology and Penology ... 0627
- Demography 0938
- Ethnic and Racial Studies 0631
- Individual and Family Studies 0628
- Industrial and Labor Relations 0629
- Public and Social Welfare 0630
- Social Structure and Development 0700
- Theory and Methods 0344

Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES
Agriculture
- General 0473
- Agronomy 0285
- Animal Culture and Nutrition 0475
- Animal Pathology 0476
- Food Science and Technology 0359
- Forestry and Wildlife 0478
- Plant Culture 0479
- Plant Pathology 0480
- Plant Physiology 0817
- Range Management 0777
- Wood Technology 0746

Biology
- General 0306
- Anatomy 0287
- Biostatistics 0308
- Botany 0309
- Cell 0379
- Ecology 0329
- Entomology 0353
- Genetics 0369
- Limnology 0793
- Microbiology 0410
- Molecular 0307
- Neuroscience 0317
- Oceanography 0416
- Physiology 0433
- Radiation 0821
- Veterinary Science 0778
- Zoology 0472

Biophysics
- General 0786
- Medical 0760

EARTH SCIENCES
Biogeochemistry 0425
Geochemistry 0996
Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES
Environmental Sciences 0768
Health Sciences
- General 0566
- Audiology 0300
- Chemotherapy 0992
- Dentistry 0567
- Education 0350
- Hospital Management 0769
- Human Development 0758
- Immunology 0982
- Medicine and Surgery 0564
- Mental Health 0347
- Nursing 0569
- Nutrition 0570
- Obstetrics and Gynecology .. 0380
- Occupational Health and Therapy 0354
- Ophthalmology 0381
- Pathology 0571
- Pharmacology 0419
- Pharmacy 0572
- Physical Therapy 0382
- Public Health 0573
- Radiology 0574
- Recreation 0575
- Speech Pathology 0460
- Toxicology 0383

Home Economics 0386

PHYSICAL SCIENCES
Pure Sciences
Chemistry
- General 0485
- Agricultural 0749
- Analytical 0486
- Biochemistry 0487
- Inorganic 0488
- Nuclear 0738
- Organic 0490
- Pharmaceutical 0491
- Physical 0494
- Polymer 0495
- Radiation 0754

Mathematics 0405
Physics
- General 0605
- Acoustics 0986
- Astronomy and Astrophysics 0606
- Atmospheric Science 0608
- Atomic 0748
- Electronics and Electricity 0607
- Elementary Particles and High Energy 0798
- Fluid and Plasma 0759
- Molecular 0609
- Nuclear 0610
- Optics 0752
- Radiation 0756
- Solid State 0611

Statistics 0463

Applied Sciences
Applied Mechanics 0346
Computer Science 0984
Engineering
- General 0537
- Aerospace 0538
- Agricultural 0539
- Automotive 0540
- Biomedical 0541
- Chemical 0542
- Civil 0543
- Electronics and Electrical 0544
- Heat and Thermodynamics ... 0348
- Hydraulic 0545
- Industrial 0546
- Marine 0547
- Materials Science 0794
- Mechanical 0548
- Metallurgy 0743
- Mining 0551
- Nuclear 0552
- Packaging 0549
- Petroleum 0765
- Sanitary and Municipal 0554
- System Science 0790

Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY
General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

THE UNIVERSITY OF MANITOBA

FACUTY OF GRADUATE STUDIES

COPYRIGHT PERMISSION

A HYPERMEDIA-BASED INTELLIGENT COMPUTER-ASSISTED INSTRUCTION MODEL

BY

OGAZI ROSE JOSHUA

A Thesis/Practicum submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am indebted to my Supervisor, Professor David H. Scuse, for his immense support, availability, suggestions and insights, and encouragement throughout my work on this thesis. I am also grateful to Professors Alfred Bork, Mark Evans, and John Allen for taking time to read the thesis and for their helpful suggestions. Thanks also go to the Canadian Commonwealth Scholarship Agency and the Federal Government of Nigeria, for offering me the scholarship that financed the research, as well as the Federal University of Technology, Owerri, for granting me study leave to enable me to carry out my studies here in Canada.

I must express unspeakable thanks to my husband, Reverend Jeremiah Joshua, for his unfailing support, encouragement and prayers offered throughout the duration of my program, and to my children, Jerry Jr., Peace, Emmanuel, and Faith, who were all born during this program, and whose arrivals made the work all the more challenging.

Above all, thanks to the Almighty God who gave me life and breath, and the strength and wisdom to carry out this research, and for providing me with a scholarship, an excellent supervisor, and all the good people who assisted me at several points of the program. Thank you, LORD, Thank you, JESUS.

ABSTRACT

A model for intelligent tutoring is discussed whose aim is to provide an adaptive system that directs a student to high-level mastery of learned concepts. Several open problems exist in intelligent tutoring, including adaptivity, sufficient subject matter, explanations, student modeling and identification and revision of beliefs. Expert-system-based tutors lack sufficient subject matter while hypermedia-based tutors lack inference knowledge and cannot provide intelligent tutoring. We have combined these two technologies for detailed study material as well as intelligent interaction with a student. Our model components include the Tutoring Strategy Module, Student Modeling Module, Expert Problem-Solver, Knowledge-base, Dynamic Questions Module, Study-Management Module, and the hypermedia component providing an electronic textbook and a browser. The adaptive and opportunistic tutor addresses misconceptions as they occur, and receives adaptation input from the Modeler. We also propose the *Collection and Invitation* intervention technique by which the tutor would not offer solution assistance to a student unless otherwise directed. Our Modeler differs from the traditional overlay and buggy models by performing solution-strategy-independent modeling based on the correctness and completeness of a student's solution. Using the final result obtained by the Expert Problem-Solver, it uses the knowledge-base to compute a set of paths to the same solution, identifies student beliefs from interactions, and identifies which paths are applied by the student based on the frequencies of beliefs in the computed strategy-base. The Modeler does not employ a mal-rule-base but labels beliefs and identifies misconceptions and missing beliefs by processing the knowledge-base. It also performs *Instructional Belief Update* by maintaining two belief sets and removing contradictions from an opposing belief-set before arriving beliefs are stored. For belief revision we have defined an improved coherencist scheme based on the resolvent model developed by Dalal (1988). We have also specified algorithms for components of the Modeler which are of reasonable running times. The Expert Problem-Solver as well as the alternative-path-computation phase of the Modeler have been implemented.

The thesis has contributed to intelligent tutoring research in three ways: solution-strategy-independent modeling, identification of misconceptions using the knowledge base rather than a mal-rule set, and the improved belief revision scheme which incorporates derived beliefs, adjusts the status of persistent beliefs whose justifiers have been deleted, and does not fail in multiple contradictions.

CHAPTER ONE: INTRODUCTION

The work presented in this thesis is centred around the age-long art of teaching, or instruction, or passing information from one (knowledgeable) source to another (assumed to be not as knowledgeable as the instructor). We refer to this as an educational or instructional process.

The educational system has undergone several stages of development. According to Bork (1981) and Goodman (1981), the earliest humans learned mainly by experience and interaction with each other. The Greek technology stepped up the learning mode by using written documents as a basis for learning. The dialog, lecture and group discussion techniques have also been employed in many societies. Dialog, demonstrated by Plato, involved one student and one instructor. The former was not lectured by the latter, but by carefully selected questions, was led to a good understanding of concepts under discussion.

Several technologies have been employed in instruction, which form the premise for this thesis. However, for a clearer understanding of the chapters herein, it is pertinent to examine, in contemporary context, what teachers, instructors or tutors (these terms are used interchangeably in this thesis) desire to achieve during the instructional process. What does an instructor want to pass across (content), and how does he want to go about doing it (methodology)? These instructional goals will help us to understand the object of the technological attempts described in subsequent chapters, as well as enable us to assess their performance effectively. The instructional goals also form the bedrock for our own design. The following section describes the processes involved in teaching, as well as the characteristics required in a good instructor.

1.1. Properties of good teaching

According to Brophy (1989, 1991, 1992), educational research until the 1970s focused on "what to teach", or pedagogic content. However, all the authors cited in this section agree that since school curricula have been established to a very large extent, research must now focus on "how to teach" or instructional methodology.

In this section, we highlight salient characteristics of good teaching, with the intent to borrow strategies to be applied in our design. Features are grouped under content and methodology headings. Although our focus is not on content, it is important to discuss some content issues, for completeness.

1.1.1. Content issues

Content issues dwell on the nature, depth and organization of the subject material to be taught. Baker & Quellmalz (1972) state that the aim of an instructor is to attract the student's attention to the instructional task, and keep it there. It is therefore necessary to meticulously organize the content of the material to motivate the student and maintain his/her interest. References on content issues are from Anderson & Roth 1989; Baker & Quellmalz 1972; Blumenfeld 1992; Brophy & Alleman, 1992; Foley & Smilansky 1980.

a) Prominent among content issues is the inclusion of an *overview*, statement of objectives, or a clear subject description. The good instructor presents a precise statement of what the student is expected to learn, and reasons for learning them. This is placed at, or near, the beginning of the material, and helps the student to stay focused on the task on hand. Moreover, the body of the material must not deviate from the stated objectives, so as not to confuse the student.

b) Related to specification of objectives, is the *clarity* of subject matter, and simplicity of presentation. These are particularly important for definitions, formulae, procedures, and other standard pieces of information which form the foundation for other learning. The good instructor avoids ambiguous statements and complex terms, since they would confuse the student and hinder learning.

c) There should be a *sequentially coherent* structure among related concepts. Easier concepts should be treated before more difficult ones. Moreover, the instructor should clearly state the transition and relationship of each concept with preceding and subsequent concepts, focusing on similarities, differences, and applications. This helps a student to create accurate conceptual links and associations, for better understanding.

d) Another feature is the *control of inspection behavior*. Students, like other people, tend to browse through reading material, paying more attention to certain features. Thus two students might read the same material, but owing to different inspection behavior and tendencies, learn different things. An instructor must therefore aim to control the student's inspection behavior positively, and draw attention to significant text features, as well as other relevant information the student must learn for subject mastery. These features are mostly text-formatting features like underlining, **bold-face**, *italicization*, UPPER CASE LETTERS, and other highlighting capabilities that tend to draw the reader's attention. Particularly, Baker & Quellmalz (1972) recommend text-embedded questions which require thought and response. These questions reduce skimming and get the student to review preceding material in order to pay more attention to certain portions of the material and to figure out correct responses. This review process ultimately reinforces learning and streamlines the student's thinking away from a broad range of concepts, to salient points addressed by the embedded questions.

e) Another content issue is the provision of *multiple examples* which would help to reinforce facts to a student, since the examples encourage viewing a concept from different perspectives, as applied to different situations. Multiple examples constitute a form of *repetition*, which serves to emphasize salient points, for the student's attention. It is also effective to provide multiple procedural strategies applicable to the same problem type.

f) Anderson & Roth (1989) state that it is more important to treat each concept to a detailed *depth*, than to teach several concepts at superficial levels. This is highly relevant because superficial knowledge of a great breadth of material does not equip a student with adequate competence to tackle practical problems in any particular domain. Learning material must therefore aim to provide adequate amounts of detail on each concept, (enough to equip a student for practical problem-solving) before proceeding to new concepts.

g) Brophy & Alleman (1992) stress the need for learning material of high cognitive levels. Concepts should neither be too simple to learn, nor too complex for a student to understand. Over-simplicity would not challenge a student, while over-complexity could be frustrating. They emphasize that although it is important for a student to understand what is taught in a learning session, the instructor must not sacrifice the applicability of learning to the real world, for simplicity. This is because if a student cannot apply what is learned in the real world, then the learning is useless. Emphasis must therefore be placed on teaching concepts that are relevant and applicable in society.

h) Just as an overview is important, it is necessary to include a *summary* of content, which ties the overall content together, and relates it to the initially defined objectives. Good conclusions allow the student to view preceding material in a coherent fashion, and comparison with initial objectives gives a sense of satisfaction and achievement.

1.1.2. Methodology issues

Methodology focuses on the techniques applied towards obtaining high levels of understanding in a student. Researchers agree that an instructor must not focus on rote learning. A student should not merely memorize facts and reproduce them on demand, but must properly understand and master concepts as well as the underlying reasoning behind them. Students tend to learn by association, building links between concepts as they learn (Anderson & Roth, 1989; Lampert, 1989). It is therefore important that an instructor presents coherent bodies of information, with clearly specified inter-relationships. It is this coherent organization that facilitates understanding and enables the student to apply conceptual learning to practical problem-solving situations. It is a general consensus that the instructor's methodology greatly influences the student's understanding and long-term achievements (Anderson & Roth 1989; Baker & Quellmalz 1972; Blumenfeld 1992; Bork 1995b; Brophy 1989, 1991, 1992; Brophy & Alleman 1992; Fennema et al., 1989; Foley and Smilansky 1980; Lampert 1989; Peterson et al., 1991) and the emphasis on current research has been on the student's *understanding*.

a) Prominent in methodology discussions is the need for *active student participation*. Active learning is preferred to passive learning, since the former keeps the student alert and interested in the learning process. Passive learning is typical of a lecture situation. Bork (1995b) reports that some students sleep in a passive learning environment while Foley & Smilansky (1980) report that a lecture audience loses attention in about 15 minutes. In a single-student instruction setting, loss of passive attention is even faster because there are fewer objects or people to look at for momentary distraction; a student who stares passively at a computer lesson tends to stray quickly into day-dreaming.

b) Another very important instructional strategy is to involve the student in *direct practice*. Practice ensures non-passive learning, since the student is required to make responses and solve problems. Secondly, it facilitates understanding, because the student is able to apply concepts and facts to different practical situations. An instructor may apply hypothetical, open-ended questions that provoke student-analysis and thought, thus enabling the student to apply procedures in "what-if" scenarios. This is related to providing the student with multiple examples, which is a content issue. However, practice differs from multiple examples in that a student can read multiple examples passively, while practice is an active process.

c) A third teaching strategy is *feedback*. For learning to be exciting and involving, the student must be given a notion of progress or otherwise. Feedback involves letting the student know of the correctness or incorrectness of responses, as well as reasons for incorrectness in the latter case. Feedback serves to avoid repetition of errors, and reduce student anxiety, and increase the student's self-esteem. If there is no feedback provided for a student who is not responding correctly to instructor stimuli, there is a high probability that the same errors would be repeated, and would ripple across to other concepts, thus hindering learning and understanding. Also, a student who is not sure of his/her progress would not be able to develop the confidence required to tackle (and may actually avoid) more complicated tasks, since there may exist some doubts as to his competence.

d) A very important technique for teaching towards understanding is *probing for student understanding and reasoning*. For an instructor to ascertain that a student has clearly understood a concept, there must be a clear indication of what the student actually understands. The instructor cannot know what the student understands without probing the student for explanations or statements that indicate depth of understanding.

Furthermore, a student may apply procedural knowledge correctly, without adequately understanding the conceptual basis behind the procedures (Lampert 1989). It is therefore pertinent for the instructor to query a student for underlying reasoning. Probing comes in the form of questions, prompts, calls for summarization, and other requests for thought-provoking student responses. A student's explanation, for example, of how a certain solution was obtained, serves to inform the instructor of the thinking process that occurred during the problem solution, as well as the facts and concepts employed by the student to obtain the solution, thus showing the form of understanding the student has acquired.

e) Individualized instruction would not be perfectly obtained if the instructor does not apply the principle of *adaptation*. This forms the heart of individualization, since it involves the instructor's adjustment of teaching styles, complexity levels, content details, and teaching speeds, based on the individual student's requirements. For instance, a novice learner learns more fundamental material, requires greater explanation details, and is generally slower than the advanced learner. Individualization is the basic reason for Computer Assisted Instruction, and optimal adaptation techniques still constitute an open research problem.

f) Closely related to adaptation is the technique of using knowledge of the student's thinking processes as a basis for instructional strategy. This is what Fennema et al. (1989) refer to as Cognitively-Guided Instruction (CGI). This goes a step beyond adaptive instruction in the sense that within the same level of instructional complexity, for instance, different students might view the same concepts from different perspectives, and think about the concepts differently. For instance, a student might view a sick patient as a malfunctioning radio-set, and the treatment process as replacing certain blown-out capacitors. Another student might view the same patient as a broken

stick, and the treatment process as nailing the pieces together. A good instructor adapts to a student's thinking, explaining the concepts using the student's own analogies, before drawing the student back to a correct perspective, when a parallel would have been obtained between the instructor's concept and the student's analogies (Lampert 1989).

g) Related to active learning and probing is the strategy of encouraging student-instructor *interaction* stressed by Bork (1987). Mere active learning and probing may be uni-directional in the sense that questions may flow only from the instructor to the student, who is required only to produce prompted facts. However, interaction suggests a *mixed-initiative* discourse, in which questions, comments and requests for change-of-direction may also come from the student, and serve to restructure the instructor's teaching schedule and style. Interaction thus departs from the traditional "teacher tells, student listens" routine. Apart from allowing students to create meanings of concepts during discourse, interaction also permits calling other related concepts into play, thus speeding up learning of individual concepts as well as building several blocks into a whole coherent piece, with their inter-relationships put into consideration. One approach to interaction is the Socratic process in which the instructor does not tell a student the answer to a question, but encourages the student to figure it out during discourse. Moreover, the student is encouraged even to answer his own questions, by an instructor asking follow-up questions. Interaction further promotes adaptation and individualization, since the instructor is made to adjust to the student's learning trend. In fact, Fennema et al. (1989) state that during interactive teaching, an instructor is continually thinking, and may make adaptive decisions as much as once every two minutes.

h) One way to measure the student's understanding and achievement is to administer a pre-test at the beginning of a learning session and a post-test at the end. The pre-test

serves to tell the instructor what the student already knows about the material to be covered and to guide the instructional process to address (pre-requisite) concepts not yet understood. The post-test serves to give the student a sense of achievement, as well as inform the instructor about how well the subject has been understood and areas to be addressed in remedial pedagogy.

i) A good instructor allows for a *variety* of problem-solving approaches. Particularly, there is no rigidity as to how a problem is to be solved. It is sufficient to prescribe standard procedures, and then to allow the student to attempt to solve problems in his or her own way. An instructor does not dismiss any new student technique as incorrect, but uses it to try to understand how the student is thinking. If the new technique is correct, the instructor must acknowledge that it is, but should suggest better approaches if the student's approach is sub-optimal. This ability to allow for variety facilitates an instructor's modeling of student understanding and promotes better adaptation to student approaches.

j) It is important for an instructor to stress the *reason* for learning. A student might be primarily concerned with obtaining good grades in a final examination or test, and thus take to memorization or other short-cuts that hinder learning (which also ultimately negatively affect the acquisition of good grades!) (Blumenfeld 1992). It is important for the instructor to stress that learning is being undertaken purely for the sake of expertise, and not for grades. This allows the student to focus more on subject mastery than on memorization.

k) Allow for post-learning *debriefing*. This is a process in which the instructor and the student review a topic or learning session, summarize what has been learned, and assess the closeness between the laid-down objectives and the learning session proper. This also serves to inform the instructor on how well the learning session was handled, as

well as suggesting areas for improvement. Debriefing differs from post-tests and summaries. A summary is part of the content, which may be text presented at the end of a major topic, while a post-test is a series of questions presented to the student to test his or her knowledge of preceding material. Debriefing is however an interactive assessment, which could include assessment of summaries as well as post-tests.

l) It is important for the instructor to *give adequate time* for a student to answer questions, without premature instructor interruption with the correct answer or with other questions. The problem of determining appropriate instructor interruption timing is still an open research problem, considering that it is sometimes difficult to predict whether or not a student will answer a question correctly, and it is impossible to allow for infinite problem-solving time.

m) A good instructor must show adequate preparedness to answer unexpected student questions. Students are usually unpredictable, and sometimes deliberately or otherwise ask questions which the instructor may not be able to answer. It is very important that an instructor studies beyond current topics being discussed in class, in order to cope comfortably with such unpredicted student actions. Where the instructor is not able to respond satisfactorily, though, s/he must show enough humility to admit ignorance and promise the students that some response will be forthcoming during the next learning session, by which time some research will have been done in that direction.

n) Other teaching strategies include good voice projection, appropriate body movement, instructor confidence, good posture, good-naturedness (smiles and heartiness). Good-naturedness enables the students to have some personal closeness to the instructor, which would free them from the fear of asking questions or expressing ideas in class. The other listed items mostly serve to draw the students' attention and prevent distraction or straying thoughts.

1.2. Thesis Layout

As mentioned earlier, several technological attempts have been made to enhance instruction. A historical account is presented in Chapter 2, along with the discussion of some educational technology devices like audio-visual aids. We will also introduce Computer-Assisted Instruction (CAI), which attempts to carry out instruction using the computer as the instructional medium. Discussions in Chapter 2 include arguments for and against CAI, and basic requirements of a CAI system.

From assessment based on good instructional goals presented in Section 1.1 above, traditional CAI does not quite measure up. Chapter 3 introduces the enhanced form of CAI, called Intelligent CAI (ICAI) or Intelligent Tutoring, which applies artificial intelligence techniques (particularly expert systems) to draw CAI closer to human performance levels. Discussions in Chapter 3 also include typical components of an ICAI system, some student modeling techniques, as well as brief descriptions of several examples of early and contemporary systems. The chapter closes by highlighting some limitations that are still present in these systems.

Chapter 4 introduces our own design for an ICAI system, which is aimed at overcoming some limitations of the systems described in Chapter 3. The hypermedia technology is introduced in this chapter, with a clear description of some of its advantages for instruction. We then describe our model architecture, and the interplay between the modules.

Chapter 5 focuses on the component of the design which are central to our research. It is impossible to fully implement all the components of an ICAI system during a time-constrained Ph.D. research. We have therefore focused detailed discussions on the student-modeling module (or Modeler). This chapter discusses parts of the Modeler

with the major emphasis being on the identification of the student's misconceptions and the development and maintenance of a consistent model of the student's beliefs.

Chapter 6 further narrows down the scope of this research and focuses on the definition of a new belief revision scheme which we employ in our model, as well as implementation algorithms for all Modeler sub-components. Examples which illustrate algorithms are presented, along with running-time estimates.

Chapter 7 concludes with comparisons between our system and some others, our contributions to knowledge, and some directions for future research. Appendices are included which give more details of problem definition, implementation approaches, and the knowledge base structure.

CHAPTER TWO: EDUCATIONAL TECHNOLOGY AND CAI

A major problem facing educators is that of a high student-instructor (SI) ratio, that is, more students than instructors. As shown in the preceding chapter, an instructor aims to have each student get excellent mastery of the subject being taught. Ideally, this is achievable in the Socratic or one-on-one instructional system, where the instructor concentrates on a single student, and provides *individualized instruction*, tailoring the contents and methods of instruction to the specific needs of that student. However, in the real society, the high SI ratio makes individualized instruction practically impossible. Educators have therefore attempted to solve this problem; this is the focus of this chapter. This chapter reviews how educational technologists have attempted to achieve their instructional goals in spite of high student-instructor (SI) ratios. The applicability of audio-visual materials and the computer are examined. Audio-visual aids and the computer technology are compared, which analysis serves as a resounding justification for Computer-Assisted Instruction.

2.1 Brief History

Over the years, the student-instructor (SI) ratio has continued to rise, following an increase in the number of people who desire to learn. Some technological aids were invented to improve on the situation. For instance, textbooks, printed in several copies, enabled students to study subjects completely. Nevertheless, the personal explanations of trained instructors were still needed. Lecturing was another attempt to contain the student explosion, but with it came the problem of loss of individualization between student and instructor, since the latter could no longer attend to each student on a one-on-one basis (Elton 1971). The tutorial discussion technique was then introduced, to break up large lecture groups into smaller units, in order to achieve a higher level of SI interaction. Unfortunately, most university tutorial groups are led by graduate students

instead of the classroom instructor, a phenomenon that introduces the added problem of students probably not getting accurate teaching from the graduate students. Today, the most common groups of learning are the lecture and tutorial discussion groups, which still deviate in different degrees from the educator's goal of individualized instruction.

2.2. The lecturing problem

Educators agree that, owing to individual differences, students differ in levels of understanding and learning speeds and styles (Alessi & Trollip 1985; Ohlsson 1987; Dunn & Harden 1971; Hills 1971). There is the slow and careful student, and there is the rapid and quick-to-grasp one (Liebmann et al., 1971). Also, there are the students who prefer abstract conceptualization, and others who prefer active experimentation (Billings & Cobb 1992). Again, some are independent while others are dependent (Larsen 1992; Douglas et al., 1988).

Lecturing involves the delivery of information to students without the expectation of significant lecturer-student interaction. Therefore slower students may fall behind in class as lectures progress. Even in smaller discussion groups, most of the time is monopolized by the more vocal students, while some others may be too shy or unwilling to express misconceptions, thus remaining poor in the subject concerned. On the other hand, if the instructor slows down to the pace of slower students, the faster ones become bored. The problem for educators, therefore, is that of evolving a balanced educational system which would facilitate subject mastery by both fast and slow students alike (Allen 1975).

2.3. Individualized learning

Owing to the different assimilation rates of students, educators saw the need for individualized learning, whereby a subject is learned at a student's own pace, until concepts are satisfactorily understood (Bork 1988; Ellis 1974). Individualized learning

removes the strain from both categories of students. The faster ones no longer need to wait until slower students have caught up with the rest of the class, and the latter can now study subjects slowly, steadily tackling problems at an individual pace. The issue, though, is how to achieve individualized learning. Wards from rich homes could enjoy the highest level of SI interaction (one instructor to one student) if the parents or guardians employ private instructors. A greater number of wards must however attend 'public' schools, where the lecturing technique is widely applied owing to the high SI ratio. How possible is individualized instruction in a public school?

A high SI ratio makes individualized instruction impossible at the lecture level. As pointed out earlier, attempts to adequately cater for slow students can result in making the faster ones idle and playful in class. The tutorial concept is one step in the right direction, because of the improved SI ratio, but it cannot be described as *individualized* learning, since the SI ratio is still not 1:1 (one-to-one).

Another situation that demands individualized instruction is a private studies in which a tutor is not present, which suggests a level below the 'ideal' SI ratio. Private students may learn by the use of a number of instructional aids, but there may still arise the need for an instructor's explanations of some concepts. Thus in spite of the fact that private study is somewhat personalized, it is still lacking in the sense that the student may have several unanswered questions.

2.4. Frames and educational technology

'Educational technology', in hardware terms, refers to communication media which can be used, alongside a teacher, textbook, and blackboard, for instructional purposes. These include audio, video or audio-visual equipment, as well as computers. In software terms, educational technology connotes the programs which drive the communication media to effect instruction. Educational technologists carry out research in human

learning and communication trends, and design processes of learning and teaching, using a combination of human and non-human resources to bring about effective instruction. In a bid to obtain individualized learning, educators evolved the use of *frames*, which are small units of instruction. A major topic is decomposed into short, logical, and easy-to-understand steps, which the student can study systematically, in order to learn the whole subject matter. This method of learning was developed by B.F. Skinner (Ellis 1974; Markle 1969). We will use the term *frame* to refer to a unit of instruction throughout the remainder of this chapter.

An instructor using frames as an instructional aid, seeks to ensure that all students understand each frame before proceeding to the next. Thus the use of frames encourages individualized learning, since each student can go through the frames at an individual pace. However, this state cannot be attained unless each student possesses a complete but separate set of frames, just like having a copy of a textbook. In cases where there is only one available set of frames (that with the instructor), which is used to demonstrate concepts before the class, the frames would be flipped over at the instructor's discretion. When this happens, fast students would flow along, but slow students would not, and this reduces to the lecturing situation. How then are frames to be most appropriately presented to achieve a reasonable level of individualization? Each student must have private and unconditional access to subject frames, so the issue of documentation comes in at this stage.

2.4.1. Audio-visual aids

Frames documented in textbook form help slower students to study at their own pace. Verbal lectures are too fast and difficult to follow for this category of students, and the use of textbooks removes problems introduced when a student forgets previous lectures which were delivered verbally. The textbook can now be used as a reference. The

textbook represents the written word, which is the lowest level of educational technology.

Graphics and sound have progressively supplanted the written word as a general conveyor of information. Venezky & Osin (1991) and Heimler et al. (1987) point out that pictures serve as retentional aids, increase a student's interest and motivation, and add appeal and more meaningfulness to otherwise barren texts. The effects of graphics are further enhanced if they are animated (Rieber & Kini 1991; Rieber et al., 1990). Audio and video equipment can be used effectively to preserve lectures so students can listen to them anywhere, and at any time. This is especially useful for private students who can receive lectures at home via audio and/or video equipment. The video technology is particularly powerful, since it combines the effects of sight and sound. Recorded lectures give students the advantage of hearing them as many times as desired, thus making for better subject mastery.

Audio-visual materials particularly facilitate learning in large class situations. The instructor faces a high risk of fatigue by an attempt to attend to every need of each student (Sanders, 1975). Students must therefore learn to use pre-recorded lectures. This point, in passing, brings out one advantage of large classes. That students can be taught using machines saves an institution the funds which would have been spent on more instructors. This is not to say that educational technology reduces the employment rate of instructors, but rather, where the latter are scarce and expensive, machines can be used to supplement the human instructor. However, instructors would still be needed to write and record lectures, and to render further explanations to students as the need may arise.

Since audio-visual lectures are well organized into topics and sub-topics, and are uniformly programmed, all students in different (sometimes remote and geographically

dispersed) locations can receive them in a standard trend. A *programmed* schedule describes the set of topics which a student must learn in sequence. A student following this schedule is said to be undergoing *programmed instruction*. This form of learning is applied in all walks of life: schools, the industry, the armed forces, for managers and the handicapped, to mention but a few.

2.5. Introducing CAI

The computer could also be viewed as an audio-visual material, and has been used world-wide to obtain high degrees of individualized instruction. The instructor provides instruction to the student via the computer, which is the medium of lecture documentation. Instructor-computer and student-computer interactions are carried out via a computer language which is designed to present lectures to the student without the instructor's intervention. Lectures are stored on disks or tapes, and recalled as many times as desired, for student use. The high memory size and availability of massive secondary storage on the computer facilitates documentation of hundreds of textbooks of information on a single computer. The computer's operating speed allows information to be retrieved from memory faster than the human reaction speed. Also, with multiprogramming and advanced communication facilities, several students are able to work simultaneously on the same lecture, stored in the same computer, via several terminals.

2.5.1. Computers in education

The involvement of the computer in the learning process is referred to in many ways. The most frequent in the literature are:

Computer-Managed Instruction (CMI),
Computer-Aided (or Assisted) Instruction (CAI),
Computer-Aided (or Assisted) Learning (CAL).

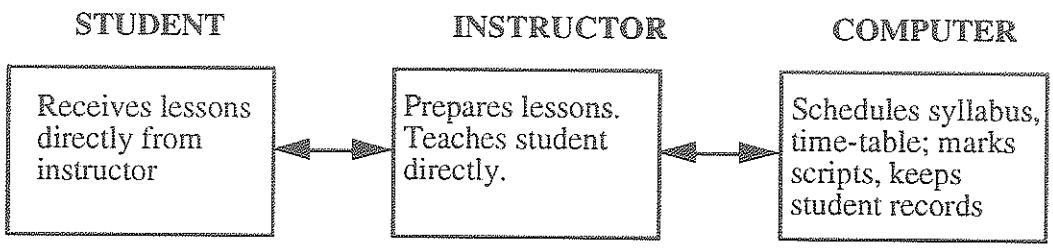


Figure 2.1: Interactions in CMI

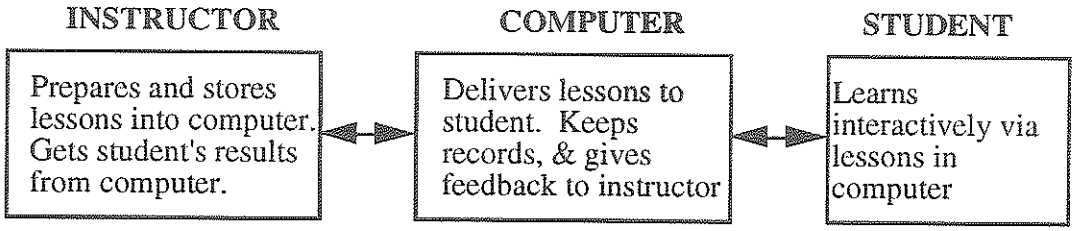


Figure 2.2: Interactions in CAI

These three terms mean essentially the same thing: using the computer to present units of instruction to a student (Barr & Feigenbaum 1982; Alessi & Trollip 1985; Ellis 1974). However, CMI involves using the computer to perform routine record keeping, or the use of the computer in the management of instruction, and has been found most useful in areas where the workload on a limited number of instructors is too high. The computer is used only to prepare lessons, schedule time-tables and syllabi, keep up-to-date record of student performance, which records would be used as a basis for the preparation of better schedules for the future. As far as students are concerned, the instructor is still the direct source of learning (Figure 2.1), and in CMI, students do not need to know that a computer is being used at all, and may in fact never come into contact with it (Broderick & Lovatt 1975).

CAI and CAL, on the other hand, mean exactly the same computer-situation, but viewed from different perspectives. The computer contains units of instruction. The instructor indirectly teaches the student using those stored units, so the student learns indirectly

from the teacher using the same units. Therefore, for the instructor, the process is Computer-Assisted Instruction, while it is Computer-Assisted Learning for the student. The computer is an assistant or aid in both cases: assisting the instructor in CAI, and the student in CAL. However, the term *CAI* (Figure 2.2) is used more often than *CAL* in the literature, mainly because most references are to projects designed to act as instructors, which is viewing the problem from the CAI perspective. The focus in our work is therefore on CAI, since our aim is to design a teaching system.

It must be stressed here that the computer cannot totally replace the instructor in the educational system. It is only an *aid* (Angibeaud 1979; Laver 1976). Thus all teaching materials must be prepared by experienced instructors, not by computer scientists. Hence as Bork (1995a) points out, efforts must be made to teach school teachers how to program the computer, towards an improvement in instructional performance. However, computer scientists are responsible for developing the software that is used to program the lessons.

2.5.2. The learning process

The learning process refers to the manner in which people obtain and assimilate knowledge. Educators agree that the best way to learn is through a *Stimulus-Response* (SR) interplay (Markle 1969; Price 1991). The stimulus is something read or heard, an input aiming at producing a response from the student. For instance, let us consider the following frame:

Q: What is 34.0*12.5/4.0?
A: _____

The stimulus is the question posed, while the response is the student-supplied answer. The SR cycle is a form of dialog, and educators propose that the interaction must be maximized in order to obtain a high level of student involvement, for effective subject

mastery. Every CAI design therefore aims at constructing instructional frames which include as much SR interaction as appropriate.

2.5.3. Frame administration

There are basically two ways in which computer frames can be presented to a student: in sequential order, or by branching.

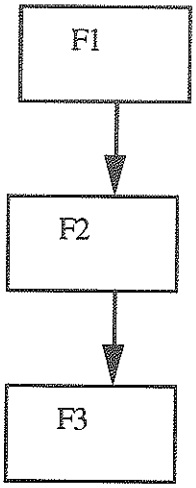


Figure 2.3: Sequential Technique

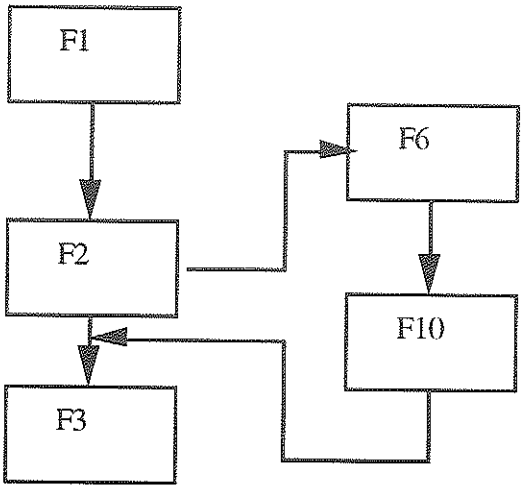


Figure 2.4: Branching Technique

The sequential form involves each student's going through the same series of frames in the same pre-programmed order, even if the order makes the topic more complicated or difficult to follow. The branching technique allows the student to branch off, where necessary, to other frames not in the defined sequence, if such branching would enhance better understanding of the subject matter. Branching also allows students to avoid studying previously known or irrelevant frames, in order to spend time on concepts which are more needed for each individual. Thus branching may produce a situation where several students studying the same topic each goes through a different frame path to achieve individual understanding, while sequential study restricts each student to the

same set and order of frames defined by the instructor. Figures 2.3 and 2.4 illustrate the two techniques.

2.5.4. Computer versus Television

Since television has been used to deliver lectures to several people, achieving some mass education, one wonders why the computer is any longer necessary for improved lecture presentation. A striking difference between the television and the computer in the educational setting is that the former is effective for *mass* instruction while the latter is for *individualized* instruction, which is the educator's ultimate goal.

Television lectures are pre-ordered on video tape and studied in sequential order, simply because the individual student at home does not have control over the communications station from which the lectures are broadcast. CAI lessons are however student-controlled and allow for branching which the student can do at will. Although branching is not automatic, but must be programmed in by the instructor or courseware author (*courseware* refers to CAI learning material), it is easily implemented in CAI, as opposed to sequential television lectures.

Television lectures are often non-responsive. The student receives instruction but cannot carry out on-line interaction with the instructor, who is not there in person. The student's questions therefore may remain unanswered, unless the instructor addresses them off-line. These questions may be brought to the instructor's attention at a later time, but the feedback is obviously not immediate in that case. However, in a live television broadcast, it is possible to wire the instructor's recording room with the students' lecture hall, and enable students to pause the instructor and ask questions during the lecture. With the computer, feedback is usually immediate and the student can pause a CAI lesson to get misconceptions straightened out before proceeding. Also, student responses are validated immediately; unacceptable answers are detected and

corrections made before the lesson continues. The student thus learns faster in CAI because of active involvement in the dialog. This is not so with the television, which may keep 'talking' without the student's attention.

Computer lessons may be reviewed as many times as desired by the CAI student, but this is not so with the television student except such lectures are recorded during transmission.

2.5.5. The interactive videodisc

Students of television lectures may record transmitted subject matter on videotape, but the wear and tear on tape is high, especially with constant rewinding to review concepts. Videotape course material is accessed mostly sequentially, and random access of specific frames is time-consuming and more difficult to obtain. The advent of the videodisc (Price 1991; Alessi & Trollip, 1991; Billings & Cobb 1992; Bork 1987; Larsen 1992; Levin 1991; Miller & Lucado 1992; Parkes & Self 1988) has added the random access dimension to video materials, taking advantage of the random access capability of discs as compared to tapes. Specific frames can be accessed randomly, and reviewed without the need to 'rewind' the disc. Wear and tear is therefore considerably less in videodiscs. We may state here also that compact discs (for example, CD-ROM) have similar advantages over cassette tapes, and therefore offer better audio services than their tape counterparts.

The term *interactive video* is sometimes used to refer to the practice of CAI in conjunction with video material. Learning is greatly enhanced since motion pictures add more illustrations to the student's learning material (Bork 1995b). Although the videodisc has an access advantage over the video tape, it is a newer technology and is more expensive, in terms of videodisc playing equipment as well as the process of producing CAI courses on videodisc. Most people may currently be able to afford

video-tape equipment, but the videodisc is a more futuristic technology, and is expected to totally replace the videotape as soon as it becomes reasonably mass-affordable.

2.5.6. CAI modes

There are several modes in which the computer may be used to deliver learning material to a student. The most commonly discussed in the literature (Rickel 1989) are the following:

- drill-and-practice
- tutorial
- simulation
- games

Drill-and-practice is a follow-up mode which serves to enable a student to memorize and practice previously learned concepts, thus transferring such knowledge from short-term to long-term memory. Examples are done repeatedly, until it is certain that a good grasp of the subject has been obtained by the student. Usually the computer selects questions randomly either from a pool provided by the instructor, or in some cases (especially in mathematics), generates random numbers to fit into question formats. The most common subject areas for which drill-and-practice has been employed are mathematics, spelling, and foreign language instruction (Price 1991).

The *tutorial* mode is the oldest CAI mode, and involves a cycle of four fundamental steps:

a) stating the item of knowledge to be taught,

b) clarifying and elaborating the point in a series of sentences,

c) asking questions to find out whether the concept has been clearly understood by the student, and

d) receiving and validating the student's response.

The tutorial mode provides a one-to-one private teacher that aims at providing new information. If properly programmed, a tutorial system can effectively relieve an

instructor of considerable teaching, and frees some time for other forms of interaction with students. Tutorial systems are self-paced, with the student in control of the speed and level of instruction presented. For instance, if a student repeatedly gives wrong answers to a frame segment, the system may assume that the subject matter is too advanced and try a lower level of instruction. The converse would be the case if the subject matter seems to be too fundamental for the student.

Simulation is a powerful CAI mode, which attempts to mimic a real-life or imaginary event or phenomenon without its attendant hazards and inconveniences (Price 1991). A ready example is a detective event which may be used to teach detectives how to apprehend a dangerous and armed criminal. Student detectives can learn this via a computer simulation of the event, rather than facing the danger of being shot or killed by a real criminal. Other events that lend themselves readily to simulations are wars, nuclear reactor environments, volcanic eruptions, air flights, medical diagnosis and treatment, and analysis of dangerous chemicals. A simulation teaches the elements of these events, in order to prepare students for tackling them when they arise in real life. A simulation student is free to explore the system, in order to discover the consequences of decisions made during the session. Such explorations enlighten the student to the different ramifications of the event under study, and provide a richer insight which can later be applied in real life.

CAI *games* enable students to learn in a fun environment. They embed concepts to be learned, and involve lots of decision making, which help students to learn things about the game domain. Common game domains are financial management, stock management and banking, in which a student practices with different circumstances, and learns the intricacies of problem-solving in these domains. The learning is mostly subtle, though, since it is done on a fun note.

2.5.7. Pros and cons of CAI

Several schools of thought have presented arguments for or against the practice of CAI. We shall discuss the cons first, since they will help us to more clearly understand the pros. Comments accompany some of the ideas, mainly to draw attention to our own views of the arguments found in the literature.

2.5.7.1. Cons

a) Kochenburger & Turcio (1974) criticize CAI for the reason that an inanimate object (the computer) cannot project the personal warmth of which a compassionate human instructor is capable. We must however point out that a human instructor also has human limits to patience and endurance. An instructor usually handles very large numbers of students, each with individual problems. The computer is therefore a welcome help, and if programmed appropriately, can project a good level of warmth in some kind language, in form of words of encouragement and praise like "Good work", "Great", "Way to go!" and so on (Price 1991). Thus for many students, the computer plays a role less than, but close to, an instructor.

b) Kayser & Coulon (1979) insist that the computer would only replace instructors and expose them to unemployment and crime. They further state that the remaining instructors would be more of classroom furniture: only there to prevent the children from destroying the computers; also that some governments may, for prestigious reasons, prefer to buy a computer rather than train an instructor.

It is true that some governments or school administrators might have preference for computer purchase over staff training, but such governments would realize in the long run, when the scarcity of qualified instructors arises, that the computer cannot write new courseware, and so would produce obsolete lessons. Also, Bork (1995a) has reported that many schools buy sophisticated computing equipment which instructors and

students do not know how to use, and which waste if appropriate instructor training is not done. Moreover, if computers are purchased with the aim to totally replace instructors, the instructional process would become completely de-personalized, depriving the student of certain insights of which only a human instructor is capable, and which may not have been programmed into the computer lessons. We therefore stress that the computer cannot be a substitute for the instructor. At best, it can be programmed to serve as a complement. An instructor need not remain idle, though, or serve as *classroom furniture*. In a CAI environment, more instructor attention should be paid to previously untaught concepts, and more thorough individual attention to students.

c) Angibeaud (1979) also argues that the computer may introduce some clumsiness into the learning process, because of its inability to think as the human would. The instructor's brain can *think around*, and cause a lesson to proceed without a break. The human mind can recall things in split times, and can generally move from one topic to another (in solving a student's problems) without making the process clumsy. Also human information can be adapted to a new environment. The computer can reproduce information, but cannot adapt to a previously unknown environment or domain, nor modify its knowledge, or introduce new ideas, without human pre-programming.

These arguments cannot be ignored. A computer may not be able to adapt to its environment, but the clumsiness in its reaction is highly reduced by its high operating speed. It is however true that, depending on the driving CAI programs, the computer may not recall all concepts that might be needed to fully meet a student's learning needs. This is where the instructor is required to offer further explanations to the student. This clearly makes the point that the computer is only an aid, and so has its limitations.

d) Price (1991) and Bork (1995a) point out that there is an insufficient number of CAI software packages in the market to cater adequately for all subject areas. Many existent software are inefficient, departing from relevant concepts, and dwelling on side attractions like sophisticated graphics or sound. Other forms of inefficiency include poor interaction language, excessive complexity for the target student population, and poor student modeling. We must state here that these, of course, are not disadvantages of CAI, but indications of poor CAI system design and implementation, and one of our research objectives is to produce a quality CAI system design.

e) Sanders (1979) states that although hardware costs have greatly reduced, prices are still relatively too high for some schools to afford. CAI would therefore drain many schools of scarce funds.

2.5.7.2. Pros

Having discussed the cons, some of the pros seem obvious, but we shall spell them out in this section.

a) One of the main problems facing an instructor of large classes is the tendency to get overworked and irritable, which adversely affects productivity. The computer helps the instructor to save energy, so as to concentrate on greater creativity, enhanced efficiency and better individual attention to students.

b) The computer has no prejudices or favorites. A human instructor is subject to prejudices which may affect communication with certain categories of students. For instance, there is a natural preference for bright and fast students, and even the best of instructors get tired of pulling the slower students along with the class. On the other hand, the computer is never tired, bored or irritable. It is 'patient' (Soloway 1992), and

if well programmed, ensures that each student understands each concept before proceeding to the next.

c) With its memory management and high operational speed, the computer is able to find, in nanoseconds, any frame in any topic, as desired by a student or instructor. Thus it offers flexible branching facilities as required. Moreover, several students can work simultaneously on different parts of the same course, because of available multiprogramming facilities.

d) The student can be actively involved in a CAI session, if it is built around the Stimulus-Response (SR) concept. Since a lesson would not continue without student input, there is a responsibility placed on the student, who would normally provide input in order to learn the concepts being taught. This is opposed to a classroom situation where an instructor may teach without some students' attention. Also, most CAI students experience a motivating psychological excitement which stems from having access to technical equipment of advanced design, and being taught by it. In fact, surveys have shown that students sometimes prefer to learn by CAI, in an impersonal and non-nagging way, rather than by human instructors (Bork 1981; Price 1991).

e) In multiprogramming environments, student-instructor interaction can be carried out via available communication facilities. This allows the student to ask questions that may not have been answered in a CAI session. When an instructor is not immediately available, student questions can be stored in an electronic mail system, so the instructor can attend to them on arrival.

f) CAI lessons are designed and authored by selected competent instructors in each field. Students are therefore saved the damage an inexperienced instructor might do by improper teaching. Such inexperienced instructors can also learn the art of designing

learning modules, by closely following the examples laid out by experienced authors in the CAI courseware.

g) A beauty of CAI is the fact that it is mostly self-controlled, enabling students to control the pace, time and sequence of learning. By being able to move at their own pace, gifted students are not bound, slower ones are not rushed, and shy ones are not embarrassed by incorrect responses given in public. Also, each student masters concepts at near-perfect levels, since CAI allows for repeated reviews.

h) There is a high level of testing in CAI, which exposes the student to questions that help to consolidate understanding. Moreover, feedback to student responses provides timely guidance, preventing students from carrying misconceptions along for too long. Also, the instructor receives reports of student errors, so that corrective instruction can be planned.

i) In courses where there is a shortage of instructors, for example, foreign languages like Russian, Chinese, or Japanese, the computer can be effectively used to teach students, making the most effective use of scarce instructors.

2.6. Requirements of a CAI system

Several suggestions have been made in the literature regarding important aspects of a CAI system (Alessi & Trollip 1991; Bork 1987; Bork et al., 1992; Heimler et al., 1987; Venezky & Osin 1991). We summarize some of these points in this section.

a) A CAI system must be easy to use, and implemented in a manner the student can understand. As much as possible, it should allow natural language interaction and not be command-oriented, in order to facilitate use by non-computer science students. If it must be command-oriented, then the commands must be easy action-connected words, and not mnemonics that may have no visible relationship to the corresponding actions.

For instance, for a student wishing to review the twentieth frame in a series, **REVIEW FRAME20** would be easier to remember than **RW 20**.

b) Basically, a CAI system is usually self-paced, allowing the student to dictate the learning speed, which is a major aspect of individualized learning. This is one of the most fundamental goals of CAI. A self-paced lesson would normally not proceed if the student has not indicated the desire to do so.

c) A CAI system should be able to identify plausible but wrong student responses, and generate corrective guidance for the student, providing as many plodding hints as appropriate. Remedial learning material should also be made available for a student who repeatedly has trouble arriving at the correct answer, to explain more fundamental issues of the concept under study. Furthermore, all students should be allowed to review lessons as many times as desired. Reviews provide students with a safe playground, permitting study without the fear of grading.

d) Graphics should be included to complement text. Graphics are very important, since they help to give the student a better mental picture of the subject matter. However, the system developer must remember that graphics tend to draw more attention than text, and so must not sacrifice learning for graphics. According to Alessi & Trollip (1991), some systems are "artistically excellent but not instructionally useful". These are systems that over-emphasize pictures, and allow students to lose sight of the real concepts being taught.

e) A CAI system should be programmed to be flexible. Particularly, some simple or common student errors, which have little or no significance to the subject, should be overlooked. An example is ignoring spelling errors, except of course in the language arts. We must add, though, that this type of flexibility may be risky. A globally

excellent student would need to be good in all curriculum subjects, and it may be more advisable to allow each subject to guard against errors which may affect mastery of other subjects. For example, language spelling errors should, at least, be pointed out in mathematics lessons.

f) Some authors suggest that a CAI system should allow for possible student modification of learning material. While this may have the advantage of presenting the concepts in a way the student would best understand, it is a most dangerous liberty because a student may delete important concepts, and even include totally erroneous ones, which would mislead other students sharing the same frames (Fiderio 1988). We therefore rather suggest that a *suggestion box* facility be provided in which a student may suggest modification ideas to an instructor, and all modifications must be made by the instructor alone.

g) A CAI system should announce its limitations. While this may give the impression of owning up to weaknesses, it is necessary, so that students and instructors can know in advance what facilities are provided. This way, there would not be unfulfilled expectations, which would be more harmful to the courseware author than if limitations were announced at the onset.

2.7. CAI languages

A language is needed to implement courseware requirements. A courseware author needs a language which facilitates the representation of all aspects of the CAI lessons, including graphics, sound, windows, different fonts, and font types and sizes. There are several language types to choose from, which are examined in this section.

2.7.1. Classes of languages

The most basic system design languages are general-purpose, high level *programming languages* like BASIC, C^{++}, PROLOG, PASCAL, and LISP. Use of a programming language by an author involves designing all desired features from scratch, using available commands and features.

Authoring languages, which are primarily designed for developing CAI lessons, are also command-oriented, but they are a level away from programming languages in that each command evokes a procedure (or some sets of procedures) which have been pre-programmed in a programming language. Authoring languages are specifically designed for non-programmers, and involve the use of specialized keywords which instructors and students use to access the system. They are not as powerful as programming languages, mainly because some effects needed by an instructor may not be included in the procedures. This state may be enhanced, however, if the instructor is allowed to modify existent procedures, or add new ones as desired. Examples of authoring languages are SuperPILOT which includes a lesson text editor, a character set editor, a graphics editor, and a sound effects editor. Others include Unison, ENGOL, and IBM's Coursewriter III.

Authoring systems are application programs or tools, which are devoid of strict programming, but help an instructor to concentrate more on instructional design. They are easy-to-use, time-saving, menu-driven systems which are like expert system shells that an instructor programs with subject knowledge. They usually have sound, graphics, and branching facilities. However, like authoring languages, they are restrictive because they impose certain limitations on an instructor, since some special effects may not be provided for in the system. Some authoring systems provide some features which may be totally irrelevant to instructors, and thus constitute memory hogs, occupying

computer memory with features that are mostly unused. System designers must therefore provide features that are relevant. Authoring systems are usually expensive (Price 1991). Examples of authoring languages are IBM's SAM IV, and Apple's *Course of Action* for Macintosh.

2.7.2. Choice of CAI language

With a range of available languages, a decision must be made as to which language to use in a design. Several criteria need to be considered to enable one to choose the most useful CAI language. Some of these are discussed below.

a) Price and transportability: High-level languages are the most fundamental of CAI languages. They are therefore the cheapest to use, especially BASIC, which comes with most personal computers. *Transportability* refers to the ability to execute a program designed on one machine to another machine of, possibly, different make. This is a major factor, considering the fact that versions of several programming languages exist across machines. It is therefore necessary to choose a language or system that would execute on as many machines as the instructor or student would possibly use.

b) Training and development time: Training time refers to the average length of time it would take for an instructor or student to learn to use a language, while development time is the length of time it would take for courseware to be designed using such a language. Training time is shortest with authoring systems, since little programming needs to be done. Development is longest with programming languages, which require a design from scratch.

c) Speed, power and flexibility: Programming languages are the fastest, most powerful and most flexible (Price 1991). They are the closest to machine language constructs, and also enable an instructor to include as many features as desired. They can therefore

be used to implement a wide range of instructional strategies like drill, practice, simulations, and games, and allow easy inclusion of graphic features.

2.8. Early CAI Systems

Early computer-based instruction applications included the PLATO system, developed at the University of Illinois in the 1960s, and the TICCIT (Time-shared Interactive Computer Controlled Instructional Television), developed in 1972 by MITRE Corporation and Brigham Young University. These were AFO (Ad-hoc-Frame Oriented) systems, which presented statically prepared texts, or 'canned' instruction. In AFO systems, the student was given a frame of author-specified text, and asked a question which required a brief response. The response was validated against pre-stored correct answers, and the student was told whether the response was right. In the event of a wrong response, the program would branch off to pre-stored remedial material. The author attempted to anticipate every wrong response, and the courseware incorporated all lessons, remedial material and instructional procedures (Barr & Feigenbaum 1982).

AFO systems failed because they depended heavily on the author's perceptions, did not allow for natural language interaction, restricted study to the instructor's syllabus and presented lessons just like textbooks. They were not flexible, had little explanation facilities, and collapsed if pre-programmed concepts were deviated from (Hume 1992).

Summary

In this chapter we presented a historical account of attempts that have been made by educational technologists towards achieving individualized instruction. We then introduced Computer Assisted Instruction along with discussion of pros, cons, and

requirements of a CAI system. We also discussed CAI languages and some of the earliest CAI systems also known as Ad-hoc Frame-Oriented or AFO systems.

In the next chapter, we examine more sophisticated CAI systems that are designed to emulate the human instructor better than AFO systems.

CHAPTER THREE: INTELLIGENT TUTORING SYSTEMS

Owing to the failure of traditional CAI or AFO systems to meet the lofty expectations of educators, research has recently been geared towards more advanced CAI programs. Artificial Intelligence (AI) techniques have therefore been applied to produce more intelligent and effective CAI systems. Natural Language Understanding makes student-computer communication more interactive, knowledge representation techniques facilitate maintenance of subject material and teaching strategies, while advanced inference methods in expert systems assist in modeling student needs.

These new systems are called Intelligent CAI (ICAI) systems, or Intelligent Tutoring Systems (ITSs) and are aimed at imitating intelligent human instructors. In these systems, course material and tutoring procedures are represented separately, so that tutoring procedures can be made applicable to different subjects. Several of these systems have also made attempts at the following ideals:

- generation of problems and remedial comments differently for each student, based on the system's student model, or perception of student performance.
- implementation of dialogs in natural language
- diagnosis of students' misunderstandings from errors made during the learning process, and
- allowing each student to follow instructional paths best suited for him/her.

ITS designers aim at producing *reactive systems*, in which the student is actively involved in tutorial dialogs and systems adapt to the student's changing needs. Presently, we describe the basic components of Intelligent Tutoring Systems.

3.1 ICAI components

There are three basic components of ICAI systems: the domain-knowledge module, the tutoring module, and the student-model module. These have been discussed by Barr &

Feigenbaum (1982), Nwana (1990a), Rickel (1989), Soloway (1992), and Tennyson & Park (1987). The need for multiple modules in a tutoring system cannot be over-emphasized. The separation of course material and tutoring procedures (system modularity), makes ITSs easier to manipulate and program. Authors agree that using several modules in a large tutoring system reduces design and programming complexity and promotes the development of domain-independent tutors that can be applied to more than one tutoring domain (Los Arcos et al., 1992; Sime & Leitch 1992; Leitch et al, 1992). Discussion of the three basic modules follows.

3.1.1 Domain-knowledge module

This module represents the subject's *knowledge-base*, and contains the information to be taught or transferred to the student. The information in this module is also used to guide the generation of questions, as well as the evaluation of the correctness of a student's solutions. Domain knowledge could be represented as any, or combinations, of the following:

- production rules: based on the IF ... THEN premise
- semantic nets: these are good for generating and answering questions involving causal or relational reasoning
- procedures: these represent the sub-skills which the student must learn, in order to fully understand the overall subject matter.

This module represents the course syllabus in data structures chosen by the system designer. However, the path followed by each student is subject to individual learning trends. Knowledge-base design is an instructor's problem which involves specification of the detailed knowledge to be taught. This would involve construction of the knowledge in order of difficulty, much as is done in textbooks or other documents, where easier concepts are presented before more difficult ones. The construction of this knowledge is a content issue, as described in Chapter One. The theoretical knowledge of any student is highly dependent on the organization and content of the domain-

knowledge module, and much care must be taken to ensure that it is as comprehensive and complete as possible, for best student mastery levels.

3.1.2 Tutoring module

This module represents tutorial algorithms and strategies, and integrates knowledge about natural language dialogs, pedagogic methods and the subject area, to optimize the learning session. This module presents dialogs, generates remedials and problems to be solved, monitors student performance, and generates comments and messages or any appropriate assistance tailored to meet the individual needs for each student.

There are different tutoring styles or models. However, good tutors are able to revise their styles if they do not fit the student's learning style. (Ohlsson 1987). The *diagnostic* model repetitively poses tasks, evaluates responses, and points out necessary corrections. The system does not suggest the solution to a problem, but allows the student to realize personal errors and to switch to better methods. The tutor keeps posing problems, steering the student along certain lines of reasoning, until the student arrives at the correct solution. This is called the *Socratic* style, and dialogs are made to resemble real-world classroom interactions. A major problem with this technique is that the tutor must say the right things at the right time, in order to lead the student to the right answer. What to say, however, depends on the system's model of the student's understanding and a faulty student model would lead to faulty tutoring. Much research is therefore being done in the area of helping the tutor to make constructive contributions.

The *discovery* model allows the student to learn in a non-formal, and not-so-restricted environment. Most games are discovery models, in which the student freely explores the application and the tutor interrupts only when a wrong line of reasoning is detected.

The *coaching* model is one in which the tutor passes instruction across without much student input.

The tutoring strategy module is a very intricate component of CAI system design, because decisions must be made, based on system-student interaction, on the next course of action to be taken for each student. Individual situations must be taken into consideration and this requires the maintenance of a *student-learning-history*. The history indicates records from previous interactions between the student and the system. Construction of student learning histories constitutes a major research area. Tied to this module is also the *expert model* which describes the system's (or instructor's) model of *some* correct response to each problem posed the student. We use the term *some correct response* because, depending on the subject domain, the expert model may not necessarily produce every correct response to a problem. Some domains (particularly non-algorithmic domains) allow for different forms of correct responses, as long as required features are present.

3.1.3 Student-model module

This module models the student's understanding of the subject area and makes hypotheses about student beliefs and/or misconceptions, so that the tutoring module can point them out, indicate why they are wrong, and suggest corrections and alternative solution strategies. The student model is based on different types of *evidence* from student information. *Implicit* evidence is derived from the student's problem-solving behavior, including preferred modes of interacting with the program. This evidence type is limited because it depends on the system's ability to recognize and describe the student's strategies. Sometimes, the system is unable to ascertain which intermediate sub-skills are known to the student, or which were wrongly applied to arrive at wrong responses. The process of identifying where exactly the student *missed it* (that is, the

exact cause of a misconception), is called the "*apportionment of credit and blame*", and has been the focus of much ITS research. *Explicit* evidence is derived from student responses to system questions, and characterizes the level of the student's understanding of the subject matter. *Historical* evidence is derived from assumptions based on the student's learning experience and goals for learning the subject, while *structural* evidence is derived from assumptions based on some measure of the difficulty of the subject matter.

The Student-modeler uses these pieces of evidence to form a model of the student's understanding. There are two major model types used by different systems in the literature. The *overlay* model assumes that the student's reasoning strategies (or solution plans) are a subset of the expert model formulated by the tutoring module, and are usually compared to those of the expert model in the same environment. The modeling component marks each skill according to whether evidence indicates that the student knows the material or not. On the other hand, the *buggy* model views the student's strategies as bugs, or deviations from the expert's knowledge. Excessive deviation from the expert's knowledge indicates that the student does not understand the concept being taught. A buggy model represents domain knowledge as rules and potential misconceptions as "mal-rules" or variants of the rules (Sleeman 1982).

Student modeling is probably the most complex part of ICAI system design, since the student does not explicitly state concepts that are not understood, but the system has to infer them from the interactions with the student. A student modeling module gets inputs from student-learning histories and constantly updates a model based on current student inputs. Student modeling also includes the construction of a *belief-base*, representing the beliefs (correct or incorrect) held by a student. This belief-base is constantly revised in what is known as a belief-revision process. Belief-revision

involves the removal of old beliefs that have been contradicted by newer student inputs, as well as the resolution of currently held beliefs that may be contradictions of one another. Belief-revision is a very salient but difficult aspect of student modeling, because the belief-base is the basis on which further tutoring is done, and an incorrectly revised belief-base would result in an incorrect tutor perception of the student's understanding, and consequently in incorrect pedagogy.

Often, it is difficult to represent a student's knowledge completely because what the system can infer depends largely on the concepts being taught. There may be several concepts which the student does not understand, but which are not recognized by the system because the interaction did not warrant the discussion of those concepts. For instance, during a French session, the system cannot determine whether or not the student understands rainfall/precipitation concepts. This is of course a comparison between two disjoint subjects, but can occur even within a single subject that involves the understanding of several intricately interwoven concepts. Thus student modeling must be restricted to things which can be explicitly deduced by the system with respect to the underlying concepts of a lesson.

3.1.4 Alternative architecture

Another ITS architecture has been proposed by Chan & Baskin (1988). This is the Learning Companion System architecture which removes an expert module and replaces it with a learning companion. Learning companion systems are based on the educational theory of peer learning (Lincoln & McAllister, 1990). We have however retained the traditional architecture because it is a better representative of the normal learning environment.

3.2. Expert systems

In the next section we describe some intelligent tutoring systems. However, first it is necessary to discuss briefly the artificial intelligence technology upon which they are built: expert systems.

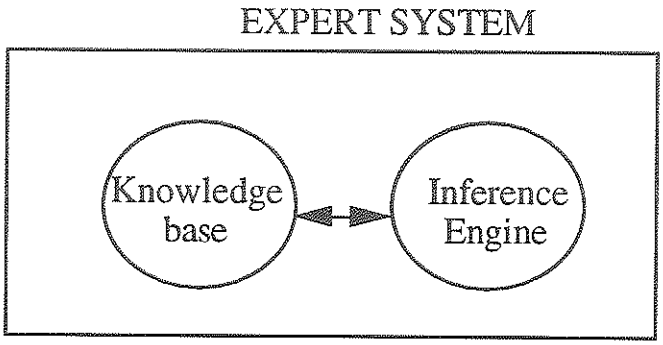


Figure 3.1: Simple schematic of an expert system

Waterman (1986) defines an expert system as *a computer program using expert knowledge to attain high levels of performance in a narrow problem area.* The key points in this definition are that expert systems incorporate high-level knowledge deemed to be obtainable from the most theoretically and practically qualified personnel in a chosen domain. These facts are stored in computer structured form and are driven by rules or procedures in an inference engine which can make intelligent deductions from the knowledge-base (see Figure 3.1).

Most expert systems are rule-based (using IF...THEN constructions) and perform forward or backward chaining. Forward chaining (or data-driven reasoning) is triggered by known events and infers new values based on available data. Backward chaining (goal-driven reasoning) attempts to verify previously postulated hypotheses which implies that the inference engine starts from a specified goal and uses an agenda to obtain information which can be used to verify the goal.

An expert system differs from conventional programs mainly in depth of knowledge, since knowledge in the system is high level expertise. Moreover, some expert systems contain what is known as *meta-knowledge*, which is *knowledge about knowledge*, or knowledge about how the system operates or reasons, which gives such systems the ability to offer functional explanations, to a good degree. This is not to say though, that expert systems have these capabilities in themselves, but as in any computer program, these features must be pre-programmed by expert-system developers. Owing to their high deductive ability, expert systems lend themselves easily to many applications in which they are used as intelligent interfaces to make inferences and generate explanations. In fact, most existing ICAI systems are built on expert systems, because of the ease with which intelligent reasoning and instruction can be done using expert systems.

3.3 Some ICAI systems

Expert systems have been used in several life application areas, including diagnosis (medical and mechanical), financial management, military arts, chemical reactor monitoring, and instruction, to mention but a few. In what follows, we shall examine some expert-system-based ICAI systems. Several such ICAI projects exist, and an attempt to discuss them exhaustively is beyond the scope of this thesis. We have therefore selected some which have served as pace-setters for many others [details are in Barr & Feigenbaum (1982) and other references]. In this section, the major features of each selected system are highlighted, with a view to assessing them and pointing out possible areas for improvement.

3.3.1 SCHOLAR

The SCHOLAR system was designed by Allan Collins and Jaime Carbonel (of Bolt, Beranek Newman Inc.) in 1970. SCHOLAR teaches South American geography, but

one of its versions, NLS-SCHOLAR, has been used to teach text-editing. Implemented in LISP, SCHOLAR is a mixed-initiative system, in which the student or tutor can direct a conversation by asking questions. The knowledge in SCHOLAR's expert module is represented using semantic nets. Each node contains properties of a geographical entity. Natural language is used to construct questions and present new material from the semantic net, which guides the interpretation of the student's responses. SCHOLAR's inference strategies for answering student questions and evaluating student responses are designed to cope with incompleteness of the knowledge-base. Items related to student questions are found by intersection search. If an item is not found, SCHOLAR responds by saying "I do not know". Below is part of a node for Peru. *SUPERC* implies "superconcept" (for example, 'PERU is a country'); *SUPERP* implies "superpart" (for example, 'PERU is part of South America').

PERU:
((EXAMPLE-NOUN PERU))
(I 0) *"Importance' of unit is high*
(SUPERC (I 0) COUNTRY)
(SUPERP (I 6) SOUTH/AMERICA) Link to superordinate units
(LOCATION (I 0) *Values of LOCATION attribute follow:*
(IN (I 0) (SOUTH/AMERICA (I 0) WESTERN))
(ON (I 0) (COAST (I 0) (OF (I 0) PACIFIC))
(LATITUDE (I 4) (RANGE (I 0) -18 0))
(LONGITUDE (I 5) (RANGE (I 0) -82 -68))
(BORDERING/COUNTRIES (I 1)
(NORTHERN (I 1) (LIST COLUMBIA ECUADOR))
(EASTERN (I 1) BRAZIL)

SCHOLAR does not maintain a student model, but relies on its Socratic tutor to diagnose student misconceptions. It then presents material which helps the student realize the error(s). Dialog continuity is weak in SCHOLAR, since it does not plan a series of questions to make a point. It therefore diagnoses errors only by following up one question with a related question.

A SCHOLAR dialog example follows. The tutor's and student's inputs are preceded by 'T' and 'S' respectively.

T: What is the language of Argentina?

S: Probably Spanish

T: Very good. Approximately what is the area of Argentina?

S: Tell me something about Peru

T: Peru is a country. It is located in South America. Its capital is Lima. Now answer the question you didn't answer before.

S: The area of Argentina is 1,100,000 square miles.

T. Approximately, but a more exact value is 1,200,000 square miles.

The tutoring module maintains an agenda which keeps track of topics being discussed. The student can ask questions that are not anticipated by the program. These are handled by the natural language facility.

3.3.2 WHY

The WHY system is an extension of SCHOLAR and was designed by Allan Collins and Albert Stevens, of Bolt, Beranek and Newman Inc. (Stevens et al., 1982). WHY teaches the causes of rainfall and concentrates on the diagnosis and correction of students' misconceptions. The WHY knowledge-base is represented in script-like data structures. For example,

First, water evaporates from an ocean or sea, then the water-laden air mass is carried over land by winds, which in turn causes the air mass to cool, which causes precipitation.

Plausible reasoning techniques are then used to extract appropriate sections for presentation to the student. Script representation however poses a problem because of some difficulty in ascertaining the correctness of a student's response which sometimes

contains correct keywords presented in an incorrect manner. The tutoring module in WHY applies Socratic heuristics to learning sessions. For example, consider the Socratic heuristics:

IF student explains causal dependence by unnecessary factors
THEN select a counterexample with the wrong value of the factor and ask why that causal dependence does not hold in that case.

Below is a dialog in which this 'Over-generalization' heuristics is applied:

T: Do you think the Andes mountain range causes heavy rainfall in the Amazon jungle?

S: Yes

T: Do you think that any place with mountains has heavy rainfall?

S: Yes

T: Southern California has mountains.
Why doesn't Southern California have heavy rainfall?

The tutor picks a counterexample for the insufficient cause, and asks the student for prior causes. The student would follow another line of reasoning if the counterexample shows that the previous one was wrong.

Like SCHOLAR, WHY does not have long-term goals for its dialogs. It asks questions on divers issues in the same session, thus finding it difficult to measure the student's understanding of each concept.

3.3.3 SOPHIE

The SOPHisticated Instructional Environment (SOPHIE) is a simulation tutor in which the student learns by trying out personal ideas, rather than by instruction (Brown et al., 1982; Yazdani 1987,). Encoded in LISP procedures, SOPHIE teaches electronic trouble-shooting and maintains an expert module which helps the student to generate, experiment with, and debug ideas.

The student is given a faulty circuit, takes a sequence of electronic measurements, and compares them with normally expected values to form hypotheses about the faults. Figure 3.2 shows the major components of SOPHIE.

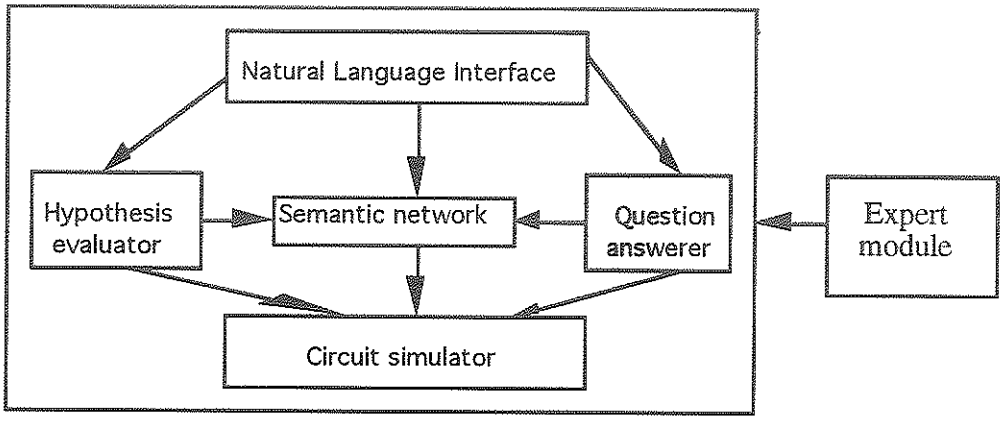


Figure 3.2. Major components of the SOPHIE system

SOPHIE's expert generates its own results, and evaluates the student's hypotheses by considering all of the information the student was expected to derive from the faulty measurements. It would then tell the student if any hypothesis is wrong, and would generate counterexamples and critiques. SOPHIE dialogs are in natural language. A dialog excerpt from SOPHIE follows:

T: This instrument has a fault. Its present settings are:

CC	*1.0*	*CURRENT CONTROL*
CR	*HIGH*	*CURRENT RANGE SWITCH*
LOAD	*20*	*LOAD RESISTANCE IN OHMS*
VC	*1.0*	*VOLTAGE CONTROL*
VR	*HIGH*	*VOLTAGE RANGE SWITCH*

S: What is the output?

T: 11.7 Volts

S: What is the output in a good instrument?

T: 19.9 Volts

S: What is the voltage between R8 and Q1?

T: 47.6 Volts

S: Replace Q6

T: Why? ... Are any terminals open?

S: Yes

T: Which terminal(s) are open?

S: The base

T: Q6 is not faulted in that manner. Q6 not replaced.

SOPHIE does not maintain a student model but a reactive environment which responds to student needs as they arise. It therefore has difficulty following up on the student's understanding, since it reacts to local needs and does not strictly follow any particular reasoning line. The student could therefore move between concepts, making the assessment process difficult.

3.3.4 WEST

WEST is a game ITS in which the student plays at a game called "*How The West Was Won*", and the program interrupts when necessary (Barr & Feigenbaum 1982; Alessi & Trollip 1985; Burton & Brown, 1982). The aim of the game control is not to give hasty corrections, so that the student can personally discover and correct system bugs without the tutor's intervention. WEST maintains an expert module, and evaluates the student's skills by comparing them with what the expert would have done in the same circumstances. The system's student model is maintained by *recognizers* and *evaluators*. Recognizers assess the student's knowledge of skills and construct a model which the evaluators interpret to determine the student's strengths and weaknesses. The coach could then make appropriate corrections, and suggest superior strategies. However, the student's model has some difficulty tracing skills used by the student. This is because WEST is a game comprising several skills and it is sometimes

difficult to recognize the student's line of reasoning from the game. Figure 3.3 shows a diagram of the WEST game system.

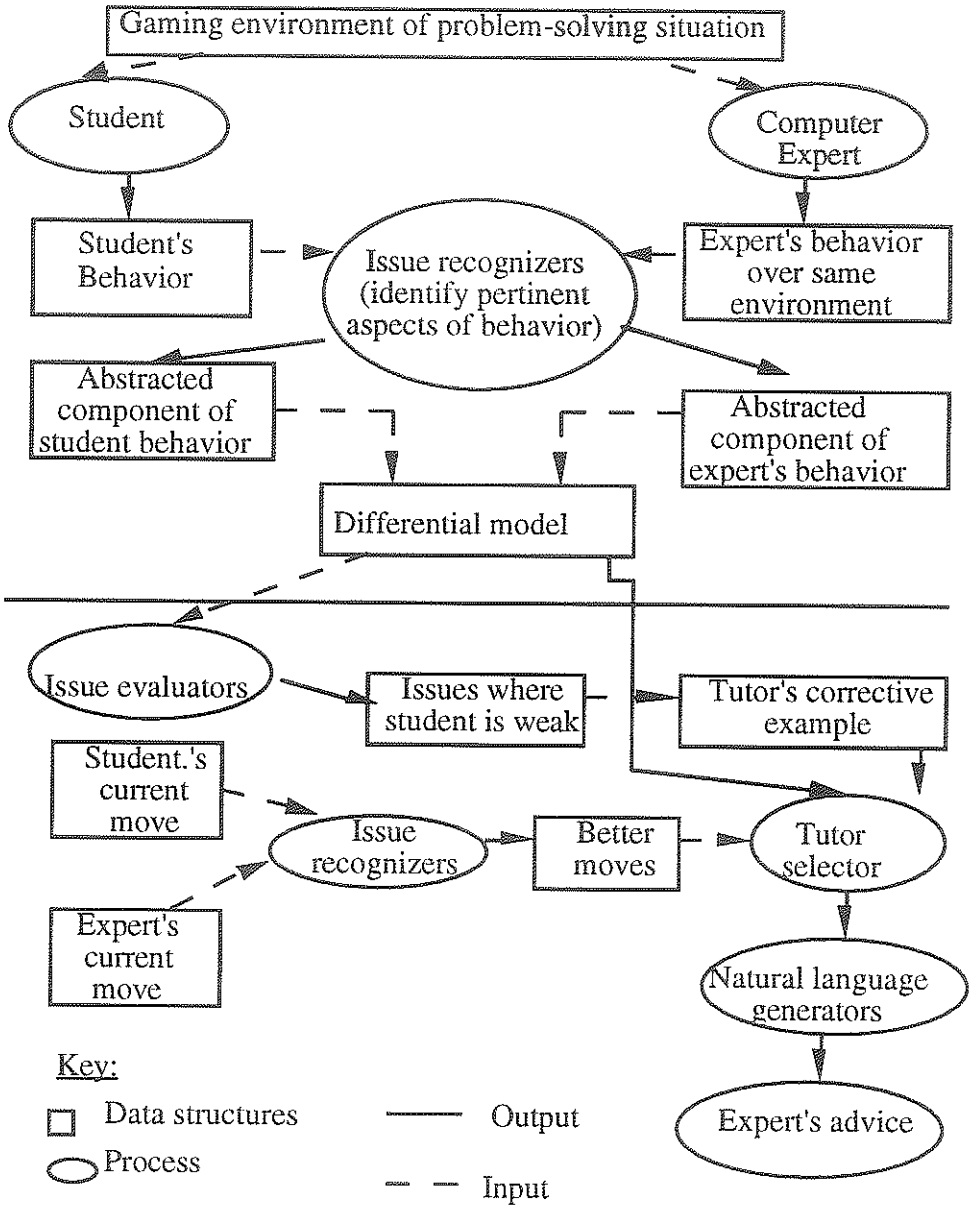


Figure 3.3. Diagram of the WEST modeling-tutoring process

3.3.5. WUMPUS

The WUMPUS system was developed by Ira Goldstein and Brian Carr at the Massachusetts Institute of Technology, in 1977. Programmed in LISP, WUMPUS is a

game in which the student aims to kill a monster in a cave while avoiding dangers like bats, falling into pits, and being eaten by the monster.

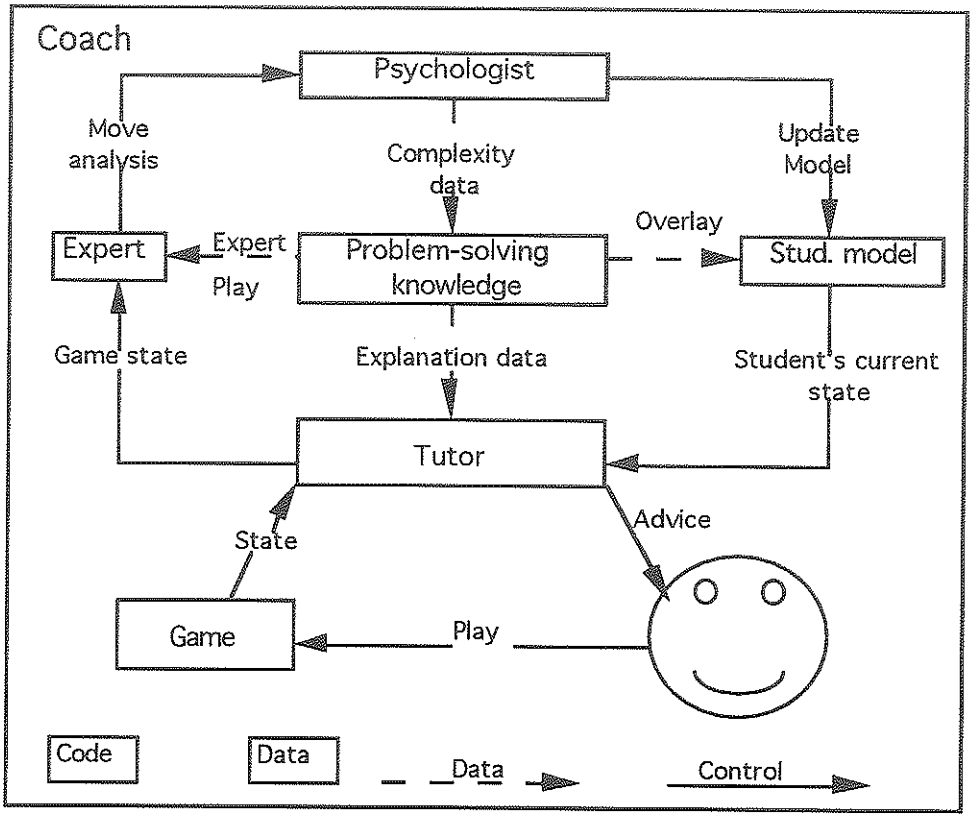


Figure 3.4. The WUMPUS computer game

WUMPUS contains four modules. The expert module, which uses a rule-based representation, informs the psychologist module of the student's non-optimal moves, and suggests better moves. The *psychologist* module then formulates hypotheses about the student's skills and stores them in the student model. The student-model (overlay) module represents the student's knowledge as a subset of the expert's knowledge. The tutoring module (called WUSOR) uses information in the student's module to guide the student to make better moves. Figure 3.4 shows the interplay of WUMPUS' components.

Examples of rules which guide the WUMPUS expert are as follows:

1) Positive evidence rule: If there is a warning in a cave, then a danger exists in a neighboring cave.

3) Double evidence rule: Multiple warnings increase the likelihood that a given cave contains a danger.

The WUMPUS expert starts the game by informing the student of neighboring caves, plus warnings about nearby dangers. An example of a WUMPUS dialog follows. The student's name is Sam.

T: Hello Sam. There are 15 caves, 2 bats, 2 pits, 1 Wumpus. You're now at cave 15 with neighbors 4,14,0. Brr! There's a draft. You are near a pit. What a stench! The Wumpus is near. What now?

S: 4

T: You are now at cave 4 with neighbors 15,14,2. Brr! Squeak! A bat is near. What now?

S: 14

T: It is not necessary to take such large risks with pits. One of caves 2 and 14 contains a pit. Also, one of caves 0 and 14 contains a pit. These constitute multiple evidence that cave 14 may contain a pit. It is less likely that cave 0 contains a pit. Hence we might want to explore cave 0 instead. Do you want to take back your move?...

The game is won by shooting an arrow at the Wumpus' lair. If all of the student's five arrows are exhausted without killing the creature, all is lost!

3.3.6 GUIDON

GUIDON is a rule-based, medical diagnostic ITS developed in the late 1970s by William Clancey at the Stanford University (Clancey 1987). Its expert module is the rule-based MYCIN, a medical diagnostic expert system. GUIDON engages the student in a dialog about a patient suspected of having an infectious disease. It then teaches the student how to get the relevant clinical data and diagnose the causative organism of the disease.

Before a session starts, GUIDON uses MYCIN to 'solve' the case to be presented to the student. The results of this background consultation are stored as an AND/OR tree of goals and rules which lead to the diagnosis. This tree is then used to evaluate the student's knowledge during the learning session. The student-model contains three inter-related components as shown in Figure 3.5: *USE-HISTORY*, *USED?* and *STUDENT-APPLIED*, each represented using certainty factors. *USE-HISTORY* is a record of rules known by the student (drawn from previous reasoning strategies, calls for system assistance and responses to questions). *STUDENT-APPLIED* is a measure of the expert's belief that a student can apply a given rule to the current case.

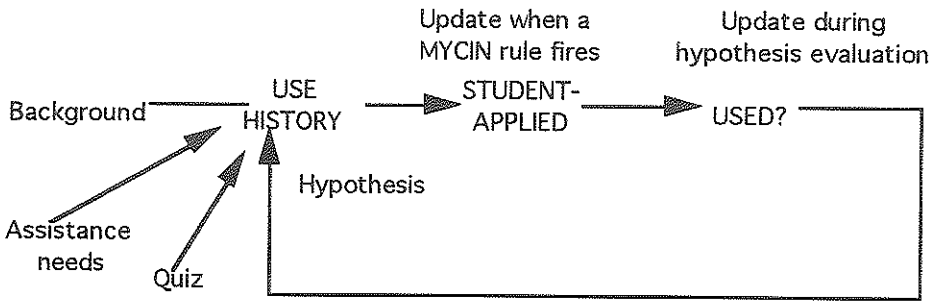


Figure 3.5. Maintenance relations for GUIDON's student-model components

Usually, knowing a rule differs from applying it. *STUDENT-APPLIED* is drawn from the student's study history. 'USED?' records the expert's belief that the student would mention a rule if asked to support a solution. It affects how the program responds to partial solutions, and feeds back into USE-HISTORY.

GUIDON's tutoring module is decomposed into several procedures, and provides keywords which simplify dialogs. Keywords include FACTORS (retrieve data which are relevant to this case), HELP or WHY(provide further information), and HYPOTHESIS (student is ready to form a hypothesis). If the student claims to have

reached a particular conclusion, the tutor probes to find out how the conclusion was arrived at. A dialog example follows.

T: What is the type of infection?

S: Bacterial

T: What facts about the case tell you the type of infection is bacterial? (Enter one factor per line)

S: PMNs

T: The percent of PMNs in the CBC is not relevant for determining the type of infection

S: PMNs in the CSF
Cerebral-fluid glucose value
Protein in the CSF

T: You're right about the CSF protein, the percent of PMNs in the CSF, and the CSF glucose value; however, you should have mentioned the fact that the meningitis is partially treated, the WBC from the CSF, and the fact that the simultaneous blood glucose value is not known...

3.3.7 BUGGY and FITS

BUGGY is an arithmetic ICAI system, designed by John Seely Brown, Richard Burton, and Kathy Larkin of Bolt, Beranek and Newman, Inc. (Burton 1982). They argue that students can completely follow their own systematic (though wrong) procedures. There is therefore a relationship between the symptoms of bugs and students' underlying misconceptions. BUGGY is used mainly to teach teachers to identify students' systematic errors in arithmetic problems. This is an interesting change from previously discussed systems which were used to instruct students. In BUGGY teachers are the target population.

BUGGY maintains a student-model which indicates possible student-misconceptions, and provides a mechanism for explaining why a student is making an arithmetical mistake, as opposed simply to identifying the mistake. It incorporates a network of sub-procedures which simulate the student's behavior and identify the parts of the student's

skills that are correct or incorrect. A procedural network with faulty sub-procedures is called a *diagnostic* model and helps the system to diagnose the student's errors. BUGGY's expert module contains correct sub-procedures, each of which is tested against the student's sub-procedures to identify possible bugs.

To train a teacher to diagnose student errors, BUGGY presents faulty solutions and supervises the teacher's bug diagnosis. Below is an example of a BUGGY dialog. The teacher's and BUGGY's inputs are preceded by "T" and "B", respectively.

B: Welcome to Bugland! Here is an example of a bug.

```
    17
  + 5
    13      Now give me problems to determine the bug.
```

```
T:    18          43        {BUGGY deliberately solves the problems
    + 6         + 79        {using the buggy procedure. The teacher
B:    15          23        {examines the cases, and claims to have
                            {diagnosed the bug.
```

T: Got the bug!

B: Here is a problem to test your theories. What is:

```
     21                     {BUGGY asks the teacher to solve a
   + 39                     {problem using the latter's suspected
T:   51                     {bug. Teacher applies it.
```

B: That's not the bug I have. Using my bug,

```
     21                     {Teacher's bug does not match, so BUGGY
   + 39                     {solves some more examples.
     15     Try giving me some more examples.
```

```
T:    51          99          68
    +1707       + 99        + 9
B:    21          36          23
```

T: Got the bug!

B: Please describe the bug you think I have.

T: The student is adding all the digits together!

B: Here are some examples to test your theories. What are:

```
      33        1091         8         28         90
    + 99      +   60     + 34      + 70       + 6
T:    24          17        15        17         15
```

B: Very good! My description of the bug is: The student always sums up all the digits, with no regards to columns.

BUGGY has been found to diagnose correctly several misconceptions in its domain.

FITS is similar to BUGGY but teaches fractions instead (Nwana & Coxhead 1988, 1989; Nwana 1990, 1991a, 1991b, 1991c, 1993). It contains a bug catalogue which represents common mal-rules in the fractions domain. This catalogue deviates from the traditional mal-rule-base structure (in which some mal-rules may not be applied at all by most students) by representing only the few mal-rules which the system designers identified as being frequent among students. Bugs in the FITS bug-catalogue are also linked to each other, allowing for the fact that one bug could lead to others. FITS performs pre-modeling (in form of pre-tests presented to the student), as well as pre-teaching required pre-requisites before main topics are attempted. Student solutions are also monitored step by step, with the tutor often providing directions to the student (not in such frequency as the model-tracing approach). The student is allowed to indicate intended strategies by selecting from a menu of actions. This menu allows the student to follow desired sequences of steps, without adhering to any previously-determined sequence. A sample FITS dialog is shown below:

T: Solve $2\frac{3}{4} + 2\frac{1}{2}$

What operation do you want to perform?
List of Operations

1. Add equivalent fractions
2. Cancel fraction
3. Find equivalent fractions
4. Sum whole numbers
5. Change improper fraction to mixed numbers
6. Change mixed numbers to improper fraction
7. Rewrite fraction
8. Provide some intermediate/final answer

WHICH <1 ... 8>?

S: 8

T: Go on then.

S: $\frac{21}{4}$

T: Correct. What operation do you want to perform?

S: 8

T: Go on then.

S: $5\frac{1}{4}$

T: Congratulations.

****

The authors of FITS have reported great success in the use of their system.

3.3.8 The Recovery Boiler Tutor (RBT)

The RBT is a simulation-based tutor in the domain of industrial training for the operators of recovery boilers (Woolf 1992). A recovery boiler is a 14-storey-high industrial structure and simple operational errors can lead to catastrophic explosions within the system. The RBT trainee is presented with a simulation of how the system operates, along with typical problem situations in which the effect(s) of the trainee's decisions and actions are immediately reflected in the system. The RBT is a very effective tutor because erroneous moves made during the simulations are avoided in real-life cases. The RBT has been successfully used to train recovery boiler operators in more than sixty paper mills in the United States of America. A sample tutor-trainee dialog follows (comments are italicized):

Problem: A partial blackout is in progress and it will lead to deterioration of the combustion process if no action is taken. Meters indicate that CO_2, SO_2, and O_2 readings are abnormally high.

[*An alarm buzzer sounds*]

T: What action are you going to take?

S: Check smelt spout cooling water. [*Student's action does not result in a change in the boiler*]

T: It looks as though the liquor is not burning ...

S: Help. What is going on? [*student requests help*]

T: A partial blackout is occurring, and this causes TRS, CO_2, SO_2, and O_2 to increase. Steam flow and steam temperature are dropping. I think part of the bed is not burning.

S: Rod the primary air ports [*A small improvement in the boiler readings can be observed*]

T: Your action is correct. However at this time rodding alone is not effective

S: Raise the liquor temperature. [*Student's action does not reverse the emergency, and a new alarm buzzer sounds*]

T: Rodding the air port apparently was not enough to undo the blackout

S: Increase the oil fuel supply. [*Student's action reverses the emergency*]

T: That solved the problem alright. Good thinking

Analysis of the problem: You had a partial blackout caused by plugged air ports and a cold bed. Partial blackout can be effectively treated through a combination of rodding the primary air ports and adding more heat. The problem can be avoided by keeping the air ports clean.
****

The RBT uses an overlay student modeler to determine errors committed by the trainee. The student's actions are measured against what an expert operator would have done, and evaluated on that basis. Also, as mentioned earlier, the trainee gets immediate evaluation of every move made, either from the simulator or from the tutor itself. Tutors similar to the RBT are the SANTURCEDEM, a boiler tutor designed in Spain under the European ESPRIT project (Los Arcos et al., 1992) and PRODS (Process Rig Demonstration System) for tutoring on heat exchange and liquid flow in an industrial process rig (Leitch et al., 1992).

3.3.9 SHERLOCK

SHERLOCK is a plan-based graphical tutor which teaches people how to link related facts (Feifer 1992). The student is presented with a graphic map containing entities like

BASEBALL-PLAYER, PITCHER, MAN, MARK-HALLWAY, or similar entity sets. The student is then expected to link the entities using links like IS-A, LEADS (that is causes, or leads to), PART (part of), EQUIV (equivalent to), PROP (property of), and NOT.

SHERLOCK contains pre-stored plans of how links are made and uses an overlay model to evaluate a student's moves based on the resident plans. If a matching plan is found for the student's entry, the system allows the move; otherwise, the move is not permitted. The system would however ask the student for reasons for that move, so that an alternative plan can be found using the new information provided by the student. The system also gives suggestions as to the right move to make in a particular situation, if the student is not making correct links.

SHERLOCK has been estimated at being 96% accurate in its evaluations, even though the developers admit that the system sometimes erroneously evaluates correct moves as incorrect, owing to the absence of the student's own plan from the system's resident plans. Moreover, the system cannot offer explanations as to why a student's moves may be incorrect.

Interestingly, there is another ITS project under the same name SHERLOCK, but it is quite different from the SHERLOCK described above. This other SHERLOCK (Lajoie & Lesgold, 1992) is a simulation-based Air Force troubleshooting trainer for F-15 manual avionics technicians.

3.3.10 The EEMT

The Electrical Equipment Maintenance Training (EEMT) system was designed in Italy, also under the European *ESPRIT* project (Bertin et al., 1992). It is simulation-based and uses an overlay student-modeler to train electrical equipment operators in an industrial

setting. A point to note here is that the system has different levels of interaction with the trainee, at different points in time. There are three interaction levels: *Tutoring*, *Coaching*, and *Self-teaching*.

The tutoring level is for novices, at which level the Tutor provides as much detail as possible to the student, pointing out every error and prescribing detailed corrections to wrong actions. Sometimes, on student request, the Tutor describes the reasoning of an expert in a similar situation. The more advanced coaching level is for students who have a good level of mastery at the job. The amount of information provided at this level is not much, and in fact, the system provides only hints and suggestions, and may not point out erroneous actions unless they would have catastrophic effects. The Self-teaching level is a yet more advanced level. The student is literally in control of what is presented. For example, the tutor does not give error remedies nor offer explanations unless the student explicitly demands such assistance. Once again, the tutor intervenes in an erroneous situation only if it would lead to catastrophic results.

The level of at which a student interacts with the Tutor is pre-specified by the student or the student's supervisor. There is therefore no dynamic transition from one level to another.

3.3.11 The STD/ITS

The STD/ITS is the Intelligent Tutoring System on Sexually Transmitted Diseases designed by Kopec et al. (1992). The developers have used a hypermedia and expert system synergy to produce an information system on AIDS, venereal warts, syphilis, herpes, gonorrhea, and chlamydia. Information on these diseases are stored in a set of HyperCard stacks called the *Smartbook*. A student simply navigates through the stacks as desired, to obtain knowledge about the diseases. There is a separate expert system module which is used to test student knowledge of the material contained in Smartbook,

as well as for explanations and response to student queries. The STD/ITS is not a diagnostic system, but simply an information dissemination and question-answering system. The developers have designed an overlay student-modeler which has not yet been implemented. The STD/ITS designers also admit that the system lacks the ability to keep track of the student's complete navigated path.

3.3.12 Some other systems

Several intelligent tutoring systems are currently in different stages of development and include CIRCSIM, an overlay- and mal-rule-based cardiovascular tutor (Hume, 1992; Woo et al., 1991; Shim et al., 1991), MACH-III, a troubleshooter (Kurland et al., 1992), and DOCENT, a system that teaches teachers to teach (Winnie & Kramer, 1988).

3.4 Some limitations

The foregoing has been a brief overview of several ICAI systems. We now attempt to evaluate these systems, with an aim to highlighting some limitations (Tennyson & Park (1987) and Sleeman & Brown (1982) have very good discussions on the limitations of ITSs) and to suggest how some of them may be overcome using Artificial Intelligence techniques. Our work in this thesis addresses several of these limitations and offers solutions that overcome several of them.

Recall that SCHOLAR, WHY and SOPHIE are locally reactive systems. Although not fully developed, the STD/ITS also falls into this category, since it cannot keep track of the student's learning path. Local reactivity makes it difficult to follow up on dialog trends and the student's understanding model. This is not desirable and ITS designers aim at keeping the student in a particular line of reasoning until desired concepts are learned. Even though ITSs are designed to satisfy the student's learning needs, this may be abused by non-serious students, on the discovery that the system loses trend as diversive questions are asked. Such students would then ask questions from one topic

to another, and learn nothing in the end. However, if the system keeps track of previous dialog paths and is able to return to points of diversion, it can be ensured that the subject matter is studied. This may be achieved by representing dialog trends with AND/OR graphs, with backtracking facilities. A tutor can therefore backtrack to points of diversion, to continue a previous dialog. This was effectively applied in GUIDON. SCHOLAR attempted at backtracking (in saying "Now answer the question you did not answer before"), but again it was only a local reaction.

It can be observed that ITSs which involve multiple skills (for example, games like WEST and WUMPUS) have difficulty with tracing which skills were known by the student. That is because a student might know a skill without applying it at expected points. This problem is somewhat difficult to solve, since no ITS can decipher a student's "mind-bugs" at any time. Furthermore, such applications are not restricted to particular lines of reasoning. However, we believe an attempt may be made by requiring the student to indicate personally known skills, which can be appropriately tested by the system. The student could also be required to indicate the skills being applied in each move. Although that may be clumsy, removing the fun from games and making them rather boring and intellectual, it would go some way to enable affected systems to keep track of skills known to students. Again, such ITSs could borrow from BUGGY's sub-procedural approach, whereby each known skill is represented as a sub-procedure, and the student's moves analyzed to ascertain which skills were used. Problem-solving in any domain generally involves the application of multiple skills, and it is the complex task of the student modeler to decipher which skills or beliefs are known to, and applied by, the student. This is not a trivial task and we shall address it in greater detail in Chapters 5 and 6.

Systems with overlay student-models are too restrictive in terms of the student's freedom to reach valid conclusions using individual approaches (Genesereth 1982). For example, GUIDON explains the student's behavior solely in terms of MYCIN's rules. This is not always desirable, because a student might reach correct conclusions, using different reasoning strategies and rules other than those in MYCIN. Most ITSs (including the RBT, SHERLOCK, PRODS, EEMT, and the STD/ITS) have this fault. We would however suggest that ITSs should promote individualization of learning, by evaluating a student's solution according to the degree of deviation from correct solutions. "Correct solutions" here implies that there may not be only a single correct solution to a problem. Several domains have several correct solutions to a problem and experts do solve problems in several different ways, so evaluation should be as strategy-independent as possible. This is the approach we have taken in our design, and we shall discuss strategy-independent evaluation in Chapters 5 and 6. Alternatively, one could argue that solution steps can be expanded from GUIDON's simple AND/OR graphs, to include all possible solution paths. The resultant graph could then be searched in comparison with the student's own solution graph. This alternative would be very difficult to implement because the overlay modeler would still expect that a student who takes a particular path must make a certain decision at a pre-programmed point-in-time, and thus follow the solution path as is specified by the expert. The problem though, is that students do not think as experts, and may actually follow a path but not make decisions in the same order as an expert would. Thus a strict overlay model would mark the student's solution as incorrect, whereas a strategy-independent model would approve of the student's solution, as long as the desired decisions were made at some point, and the correct solution was obtained. Thus it is not the "how" (procedure) of the solution, but the "what" (content of solution) that is being evaluated, and the overall aim here is to encourage students to figure out their own solution strategies.

Probably the most common limitation of older ITSs is the near-total absence of graphics in dialogs. Apart from the electronic circuit drawings in SOPHIE, most other ITSs rarely use graphical displays. It has however been established that students learn better with graphic representations than with texts (Alessi & Trollip 1985), and ITS designers should incorporate extensive graphics. Some of the graphics limitations have however been overcome by newer ITSs (like the STD/ITS) which use hypermedia environments, and can therefore incorporate animation, motion video, and sound.

ITSs should also be designed to provide adequate help and explanation facilities. GUIDON offers some help, but they are structured in terms of MYCIN rules, which sometimes need further (unavailable) explanations.

Each system described above was programmed from scratch, using a programming language (usually LISP). With the advent of expert systems and knowledge engineering tools, we would suggest that ITS designers attempt to implement tutoring systems on ready-made authoring systems, instead of performing program development from scratch. Authoring systems have most of their programming done, leaving the ITS designer to fill in the provided blanks (Sleeman 1982). A problem with authoring languages, though, is that they may not have all the facilities needed to efficiently program some features of ITSs, particularly in complex graphical or scientific domains. In such cases, it would still be more effective for designers to program from scratch, in spite of the greater cost in time that the alternative would involve.

The computer functions only within the structure of its knowledge base. Even natural language processing is still restricted to knowledge bases. For instance, ITSs cannot generate or understand dialogs for newly-encountered learning terms. Hence, interaction is still restricted to specifically structured 'natural language'. It would take

more years of research in natural language understanding and processing before this problem is totally eliminated.

We believe that the current flaws in tutoring procedures are due mostly to the fact that ITS designers do not often consult with educational psychologists to ascertain how best to impart learning material to students. Since ITS is a marriage of artificial intelligence and educational techniques, ITS designers should seek advice from instructional experts, as their input would greatly enhance the performance of tutoring systems (Sleeman 1982).

Usually, ITSs should be able to critique how closely the student model predicts the student's behavior. For instance, extreme reasoning inconsistency or an unexpected demonstration of expertise might indicate that the program's representation does not quite capture the student's approach. It would also be helpful if the modeling process could attempt to measure whether or not the student is actually learning (Lawler & Yazdani 1987; Frye et al., 1988), and to discern what teaching methods are most effective for the student. For example, if the student's responses are all wrong, it is highly probable that the subject matter is too advanced and a lower level of knowledge would need to be taught. This is a student modeling problem, and is tied to adaptation to the student's cognitive needs. We have addressed that problem, which is a very central theme of our tutoring module.

Most current ITSs have focused on limited topics, resulting in task-specific systems, with little generalizability. Some systems claim generalizability but this cannot be without modifications (which would again result in task-specific systems). For example, NLS-SCHOLAR (a version of SCHOLAR) was used to teach text-editing, a deviation from SCHOLAR's geography. However, NLS-SCHOLAR is also task-specific to text editing That is probably the reason most ITSs are not commercialized.

There is therefore the need to design generalized tutoring systems which can be used across subjects. It is a long-term research goal which we believe is worthwhile.

The field of Intelligent Tutoring Systems is a growing one which will require many years of further research. One looks forward to a situation where ITSs will be intelligent enough to teach subjects at the right level of details, evaluate and understand the student's idiosyncrasies and solutions accurately, suggest the right issue at the right time, provide adequate help/explanations, allow for student's expression in unrestricted natural language, and be generally optimized for each student.

In the next chapter, we present general framework of our model, taking the foregoing discussion into consideration.

CHAPTER FOUR: COMPONENTS OF OUR MODEL

This chapter discusses our model which is a marriage of expert system and hypermedia technologies [what Rada (1991) refers to as *expertext*] in an intelligent tutoring environment. We start with a description of the hypermedia technology and its applicability to the intelligent tutoring problem. We then describe our model which is composed of a hypermedia unit, an expert-system-based Expert Problem-Solver, a knowledge-base, a Dynamic Questions Module, a Study Management Module, a Student Modeling Module, a Teaching Strategy Module, and an Interface. The Interface refers to the user environment including graphics modules that may facilitate student-system communication, but is outside the scope of this research. Each of the other components is then described in terms of its theoretical functionality.

4.1 Hypertext/Hypermedia

4.1.1 A definition

The word *hypermedia* is a derivative of *hypertext*. Several definitions have been proffered for "hypertext" in the literature, but we shall simply define that term using the key elements involved.

Hypertext is an environment which involves the organization of a large document into small separable units which may be interconnected using typed links. The links attempt to assemble units (called cards or frames or notecards, and so on) into larger, usually hierarchical units, which tend to present the same idea in a continuous and coherent manner. The hierarchical organization of nodes enables a user to view information at different levels of abstraction or detail.

Hypertext is a software environment based on the associative linking of related objects (Price 1991; Rada 1991). Association helps the human brain to store and retrieve

information quickly and intuitively. For instance, one learns to associate the white cross sign to hospitals, and the maple leaf to Canada. Such associations enable one to make fast analogies and to think creatively.

Hypertext software permits courseware authors to relate different types of pieces of information using associative links. Hypertext originally referred to textual information, but can be extended to include graphics, audio, video, and animation, to give a *hypermedia* environment, which connotes using hypertext alongside multimedia systems like CD-ROM or interactive videodiscs. Sometimes though, the two terms (hypertext and hypermedia) are used interchangeably in the literature (Smith & Weiss 1988; Akscyn et al., 1988; Halasz 1988; Shneiderman 1992).

4.1.2 Features of a Hypermedia system

Basically, a hypermedia system consists of the following entities:

- nodes
- links

Hypermedia systems do not use standard database record structures but screen-size units called nodes. A node may be viewed as a frame, except that a node may consist of one or more frames. A node typically consists of a node name, title, body, comments, and other slots that differ across system designers.

A link can be viewed as a cross-reference or graphical link similar to internodal links in a semantic network. Links join nodes that are related in some way and may be typed depending on the nature of the nodes they connect. Simple examples are text links and graphic links. Links may be icons or text, with attached procedures which fire when appropriate links are activated. Nodes are linked to form hierarchical or non-hierarchical structures. Hierarchical structures permit better branching. The links assist in navigating through the learning material and may be identified by icons or buttons

which a student indicates for movement in any direction. For instance links could be named NEXT, PREVIOUS, FIRST, LAST, etc. A student chooses a desired node by selecting the corresponding link.

Nodes can be combined to form compound nodes or *composites* (Halasz, 1988). A composite can be viewed as a single entity, and operations on a parent composite simultaneously affects all constituent nodes. A single node may occur in multiple composites. Consequently, all updates to such nodes must be reflected on each parent composite. The update problem is a database issue and we shall not attempt to dwell upon it in this work.

Hypermedia is a browsing environment, which implies that nodes can be grouped or linked together into plausible sub-networks through which the user navigates by following links from node to node, as desired. Browsers are usually provided which give an overview of nodes to be navigated. A user may edit nodes in an editing mode. However, in some systems like KMS (Akscyn et al., 1988), there is no difference between editing and navigation modes, and a user may edit nodes and links during navigation.

Hypermedia differs from conventional paper documents in that it is non-linear and flexible in the sense that nodes can be visited linearly or otherwise. If present in a network, a hypermedia system can fetch sequences from remote stations. Hypermedia documents are easier to edit than textbooks, owing to computer editing facilities.

Hypermedia systems are not programming languages but more of authoring systems which permit the generation of frames containing text, graphics, or animation. They are however simpler to use than authoring systems because the basic steps involved are the design and storage of concepts into nodes, and the linkage of nodes in a logical order.

In addition, a very important aspect of hypermedia is the ability to include sound, motion video, animation, and general advanced graphics features (Bergeron & Paquette 1988; Rieber & Kini 1991). This is why hypermedia is sometimes referred to as *multimedia*, depicting the inclusion of different types of information representations in the same application (although multimedia does not imply the ability to browse).

4.1.3 Usefulness of hypermedia for CAI

Since a hypermedia system basically constitutes a browsing environment, it lends itself effectively to Computer-Assisted Instruction.

Fundamentally, a hypermedia-oriented CAI system should consist of learning material stored as nodes by an instructor (usually called an author) and linked in such a way that a student can effectively obtain the desired knowledge from the system. The student would learn by navigating through the nodes, either in a sequence pre-determined by the instructor, or in a student-directed sequence, or a mixture of both (Bork 1988, 1995b; Hartman et al., 1992; Lanza & Roselli 1991; Levin 1991; Redish et al., 1992; Rieber & Kini 1990; Rieber et al., 1991). In any case, the system would contain an underlying tutoring engine which should direct the student-system interaction. Although this sounds quite simple, the issues of node structure, links, and student-navigation styles, have generated much controversy with respect to tutoring.

At this point, it must be noted that the hypermedia model is a good tool for the storage of CAI material. It can serve as a learning knowledge-base which holds all the text, graphics, et cetera, required to educate the student. For example, the QUICK system is a hypermedia-based electronic textbook of internal medicine which has been found to be very useful in instruction (First et al., 1985).

The ability to include any desired form of information representation makes hypermedia very useful in intelligent tutoring. This is because an instructor can include real-life motion video slides in a learning package and the student is able to view them on-line instead of imagining how the video processes work. An instructor can also include sound effects; for example, actually speaking every word that is being taught. These features make learning more interesting and effective.

4.1.4 Some shortcomings of hypermedia systems

Owing mostly to its simple node, link, and browse (NLB) structure, the hypermedia technology currently faces some inherent drawbacks. The most frequently cited in the literature are discussed in this section, along with some suggestions which have been made towards their alleviation. Refer to Akscyn et al. (1988) and Halasz (1988) for in-depth discussion of hypermedia issues.

An attempt to solve all the issues raised in this section would require a great deal of database research which is outside the scope of this work. However, we shall bear these limitations in mind in our design and aim at least to minimize some of the problems as much as possible.

4.1.4.1 Search and query

As pointed out earlier, a hypermedia user gets information by navigating from node to node. Often, a user gets lost in the mesh of nodes and may not find the desired information. Sometimes too, a user has to navigate through unwanted information just to get at desired nodes. To minimize the consequent user frustration, it is very important to have a query facility in or over a hypermedia system, through which a user can directly access nodes, rather than a restricted navigational process. Such query processor would produce matching nodes or sub-networks, for easier navigation or examination by the user. Queries could be by nodal content (for example, *all nodes*

with information on Sine Law) or by structure (for example, *all sub-networks containing any node on Boyle's Law or Charles's Law*). Such improved system would keep an index or other appropriate track of the contents of each node in the entire *hyperbase* (a term used for a hypermedia document, or a large collection of related nodes). Such facilities would naturally increase the memory requirements of the system. In addition, Akscyn et al. (1988) suggest using extra facilities like *HOME, GOTO* links which can assist a disoriented user to return to familiar grounds.

4.1.4.2 Dynamic update

Current hypermedia systems are pre-organized and static. This means that node contents are structured by the author and changes are not reflected unless the network is explicitly edited by the author. Thus hypermedia systems do not update the information they contain, mostly because they cannot reason about their contents (no deductive ability). This is not desirable, since information can change frequently and an author may accidentally omit the manual update of some key nodes. There is therefore the need to provide a mechanism whereby the system can dynamically update its nodes and links based on user queries. This would save update time and make the system more easily adaptable to changing information.

4.1.4.3 Versioning

The issue of versioning also arises in a hypermedia environment. It is important to maintain a history of changes made to a network, since such records help a user to have an overview of how salient ideas evolved. Furthermore, in cases where several users are working on the same document (collaborative work), it is necessary to keep separate versions of the same network, as viewed by separate users, as well as prevent inter-version interference. In a multi-version environment, the query facility must be enhanced to allow for the retrieval of any specified version and not necessarily the latest

one. Multiple versions of documents also facilitate information retrieval in cases of system breakdown or failure.

4.1.4.4 Computation

Computation refers to the active derivation of new information, as well as the ability of a system to store such derived information dynamically. The latter part relates to dynamic update discussed in 4.1.4.2 above. Dynamic computation is a key element in any CAI system. This is because during learning sessions, the system must dynamically receive student's responses, compute scores, update student models, and determine subsequent instructional trends. Hypermedia systems are currently not well equipped for these features, owing mainly to the fact that they are inherently passive storage and retrieval systems. There is therefore the need to upgrade the hypermedia model to include a computational engine which would be able to manipulate dynamic information in the system, without user intervention.

4.1.5 Assumptions on hypermedia issues

Most of the issues raised above are database issues, since they deal with the storage and retrieval of information. Related concerns are access rights, especially in a collaborative environment, plus concurrency control and communication. We shall therefore not attempt to solve these problems with respect to hypermedia systems, but assume that our hypermedia model is complete and functions efficiently with respect to database issues.

Owing however to the nature of this work, in which an expert system layer is linked with a hypermedia layer, we shall minimize some of the shortcomings mentioned above by using an expert system as a computational, query, and inference engine.

Examples of hypermedia products include NoteCards, Guide, Intermedia, Neptune, HyperStudio and HyperScreen, Apple's Tutor-Tech and HyperCard, and IBM's

Linkway and SuperCard. One of the most prominent is Apple's HyperCard, which uses a *card* metaphor, each card equivalent to a node, and several linked cards forming a *stack*, with each stack having a separate organizer or stack controller. Most of these hypertext/hypermedia systems include advanced text and graphics editors.

Hypermedia systems permit high levels of student control, allowing the student to explore a learning system, navigating through nodes to suit personal cognitive needs (Boone & Higgins 1991; Ambrose 1991). This has however been condemned for the simple reason that a student may navigate through irrelevant or non-core nodes, therefore losing the knowledge intended by the instructor (Fiderio 1988; Park 1991; Ambrose 1991; van den Berg & Watt 1991). As much as possible therefore, student-control should be more restricted. A major problem with instructional hypermedia systems therefore is the question of how to connect nodes without *semantic disconnection* (Park 1991). Semantic disconnection refers to the situation where a hypermedia user may lose grip of a subject being studied by navigating into nodes containing irrelevant or unrelated material. This is very relevant because of multi-path facilities available to students. An instructor can minimize semantic disconnection by connecting nodes in such a way that each student gets the core of the subject matter, irrespective of navigated paths.

4.2 Our model

A schematic diagram of our model is shown in Figure 4.1. Basically, the hypermedia component would contain the domain knowledge base, while the expert system contains

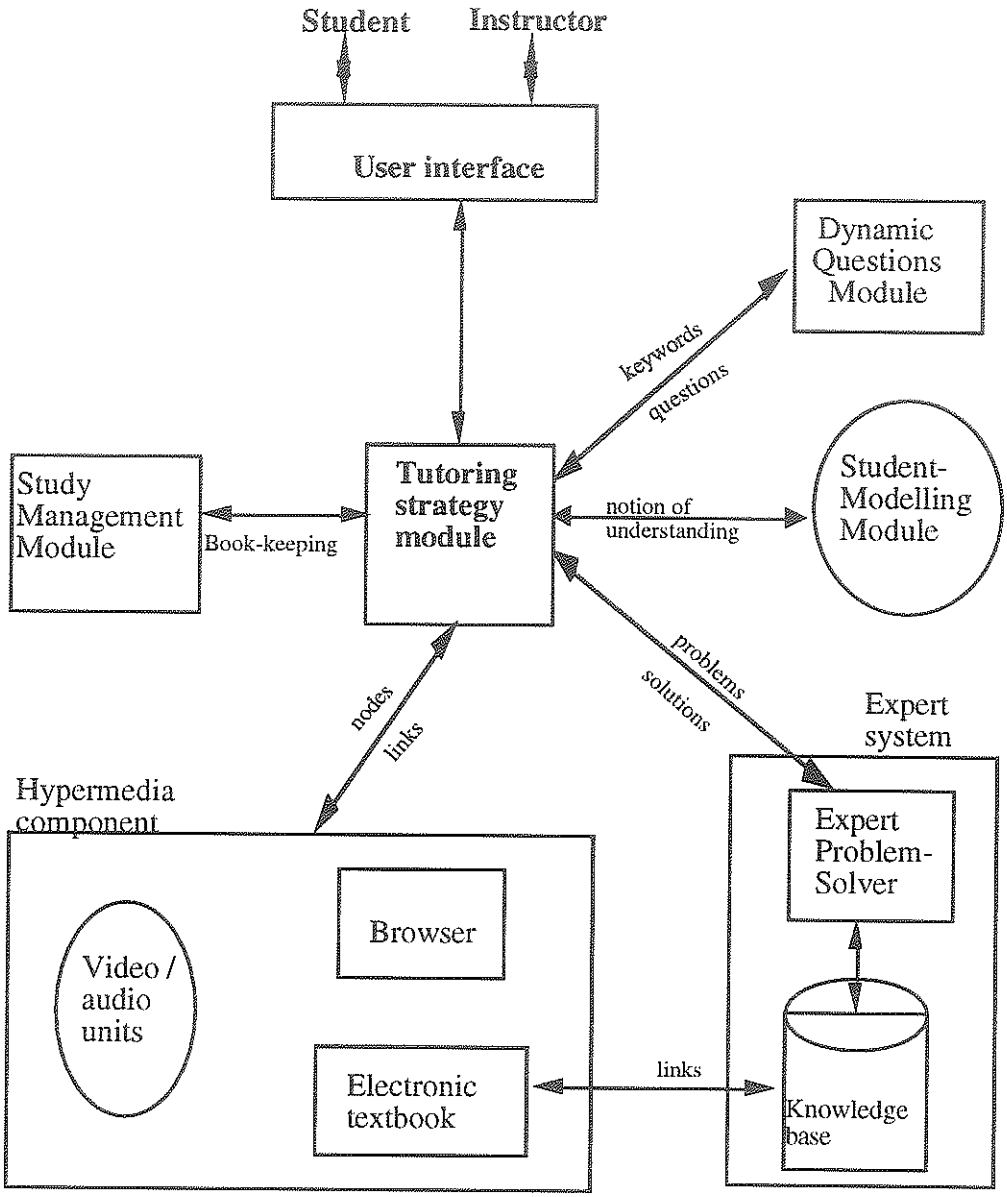


Figure 4.1: Diagram of our tutoring model

the intelligent interface. The *User Interface* in the diagram depicts computer facilities like natural language, graphics user interfaces, and simultaneous multiple-window facilities, which we assume are present in the user environment since they are outside the scope of this project. Note that the *Expert Problem-Solver* differs from the *Knowledge Base,* as explained in Section 3.1. Our emphasis in this work will be on the student

modeling module. The tutoring-strategy and student modeling modules are very closely linked together in the sense that apart from the basic knowledge-transmission cycle, the tutor cannot determine what next to teach until some input is obtained from the student and until the system has some idea what form of instruction the student requires. We shall now discuss issues surrounding this design.

4.3 Theoretical Functionality

Much of the required theoretical functionality of our model has been discussed in previous chapters, but we shall highlight them in this section.

The *hypermedia component* comprises three sub-modules: the electronic textbook, the browser, and the video/audio units. The *video/audio units* are external features attached to the computer system to provide natural motion pictures and sound via videodiscs or sound systems like tape recorders, to give a true hypermedia environment. The *browser* provides hierarchical overviews or summaries of nodes in the electronic textbook, depicting the nodal links in a block tree form. Nodes are organized in levels of abstractions, and clicking on a node at a high abstraction level reveals the next level of details of that node, and so on, until the lowest abstraction level is reached. The electronic *textbook* contains learning material comprising of text, graphics, or animation. This material is structured by the subject instructor or author, using a hypermedia application. The authoring process requires breaking down the knowledge into several nodes (sometimes in the order of thousands, depending on the size of the material) and defining links between them. We believe that the construction of a hypermedia CAI knowledge-base is not an easy task. Particularly challenging is the definition of nodal interconnections, which greatly influence navigation through the system, and consequently the net amount of knowledge passed across to the student. Some authors suggest that links be strictly pre-structured by the instructor and the student be obliged

to follow such prescribed knowledge. Others suggest that students be allowed to navigate through the knowledge base without the instructor's supervision. In the former case, an instructor is faced with the task of constructing nodal linkages for optimal learning, while the latter case requires little or no link definitions, since a student would specify links as personally desired. Our view is a compromise between the two schools of thought: allow the instructor to specify optimal learning paths, and allow the student freedom to choose between equally likely nodes. This means that if we have a structure like that shown in Figure 4.2, where *Topic C* must be learned after Topic A, but in any order alongside Topic B, the student is obligated to complete Topic A before going on to Topic C, but may choose to study Topic B before Topic C. In this scenario, the following sequences may be allowed:

Topic A -> Topic B -> Topic C
Topic B -> Topic A -> Topic C
Topic A -> Topic C -> Topic B

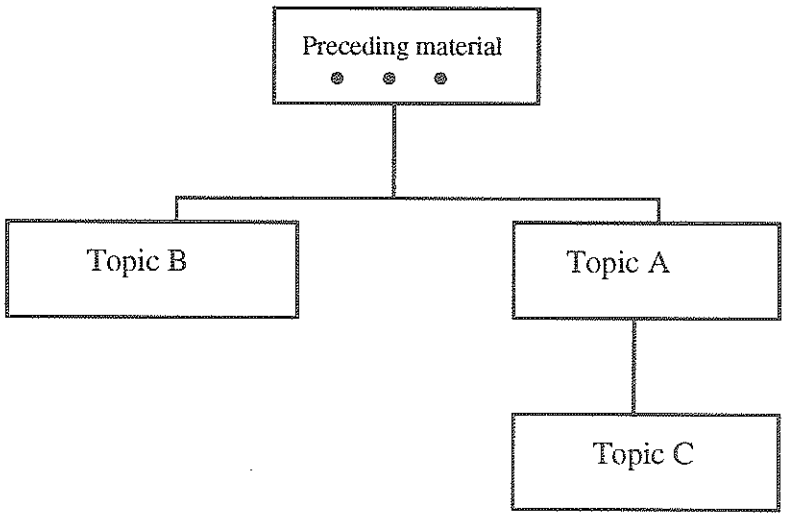


Figure 4.2: Alternative learning paths

while the following sequences are not allowed:

Topic C -> Topic A-> Topic B
Topic B-> Topic C -> Topic A

An instructor knows the knowledge a student is required to have obtained by the end of a course and so should specify guidelines and nodal links towards the acquisition of such knowledge. However, since our emphasis is not on the electronic textbook, we shall assume that the hypermedia component is properly structured. We shall be concerned only with retrieving and manipulating its contents to suit our strategies.

Overall, the hypermedia system constitutes the heart of our design's store of fundamental learning material, and we summarize its functions as follows:

- provides an overview or hierarchical structure of learning material
- provides data in multiple representations: text, graphics, animation
- provides a user-friendly environment which allows easy choice of options by single mouse-clicks
- provides well structured chunks or nodes of learning material
- provides a means for linking nodes to facilitate conceptual understanding by systematic access of related information
- allows the student some degree of freedom to choose a learning path
- provides icons for multi-directional views of information, as well as several study options such as SUSPEND, RESUME, QUIT, REVISE
- communicates directly with the Tutor, and the knowledge base (the knowledge base contains link information to nodes from which its rules are constructed)

The *Expert Problem-Solver* (EPS) is a rule-based expert system which is used by the tutor as a model with which a student's responses are compared. It answers questions dynamically during learning, and the complexity of each question is measured by the number of rules and/or steps applied by the EPS to solve any given problem. It communicates indirectly (through the knowledge base) with the electronic textbook, to offer explanations to the students on steps taken to arrive at, or reasons behind, certain problem solutions. As stated earlier, the knowledge base contains link information to the hypertext nodes from which its rules are constructed. When explanations are needed by the student, the EPS obtains the desired links from the knowledge base and passes the links on to the Tutor for retrieval from the textbook. Since the aim of instruction is to enable a student to solve problems with expertise, it is natural to assess

the student's performance with the help of a model solution (seen from a domain expert's viewpoint) for each problem. The Expert Problem Solver's solution is however not matched strictly with the student's solution but only serves as a guide to the correct evaluation of the student's solution. This will be explained further in the discussion of our student modeler We have separated the expert problem-solving unit from the electronic textbook for two reasons:

- First, for any single problem or question encountered during learning, the fraction of the textbook in which the solution(s) can be found is very minute. Owing to the length of time it would take the system to search the entire textbook each time, it is wiser to extract such relevant portions into the expert module, to save processing time.

- Secondly, contents of the electronic textbook are represented in natural language. This is an unformalized representation in the sense that a specialized Natural Language Processor would be needed to decipher solutions to questions from a body of unstructured text. Since natural language research is outside the scope of this project, we have structured the knowledge represented in the electronic textbook into a rule-based knowledge module which will be referenced by the expert problem-solver for solution extraction purposes.

A summary of the functions of the Expert Problem Solver is as follows:

- serves as a model of expert reasoning
- receives the same questions posed to the student
- applies rules in the knowledge base to solve problems dynamically
- returns correct responses to the Tutor for pedagogic use
- offers explanations for responses and reasoning strategies
- communicates directly with the knowledge base and Tutor, and indirectly with the Dynamic Questions Module via the Tutor

Ideally, the Expert Problem-Solver should obtain solutions directly from the electronic textbook, but that would involve a mechanism for composing rules from a body of

unstructured text. Since this is still an open research problem, we have taken the option of constructing the *Knowledge Base* by hand, but links are included which point to the portions of the electronic textbook from which the knowledge rules were obtained. These links are therefore accessed whenever the expert needs to offer explanations to the student. The Knowledge Base is constructed in an expert system environment and its conceptual basis is summarized as follows:

- contains rules which lend expert system structure to unformalized data from the electronic textbook
- enables the Expert Problem-Solver to deduce solutions to problems posed by the Dynamic Questions Module
- contains information on the appropriate links to nodes in the electronic textbook from which its rules are constructed
- locates from the electronic textbook the appropriate nodes needed for explanation purposes
- outputs all acquired information and results to the Expert Problem-Solver.
- communicates with the Expert Problem-Solver and the electronic textbook

The *Dynamic Questions Module* (DQM) is used by the Tutor to pose questions to the student during the learning process. Early CAI systems contained canned questions which were embedded into the learning text or were taken from a pool during learning. It is however more efficient to pose dynamic questions to suit the level of a student's understanding. In our design, this module contains the schema or framework for questions in the learning domain. These schema are organized in levels of complexity. The tutor only supplies keywords or data for the schema, and the DQM produces dynamic questions judging from the student's level of understanding. This provides an environment for a *what-if* scenario, since the tutor can alter certain parameters to produce different questions, using the same schema.

The DQM thus plays a central role in all question-answering phases, particularly the problem-solving phase of learning. During this process, the Tutor collects questions from the DQM, poses to the student, and then continues with pedagogic guidance. A summary of the DQM's functions is as follows:

- contains question frameworks or schema
- receives keywords from the Tutor
- poses questions dynamically
- organizes problems in levels of complexity
- provides a "what-if" experimental environment

The *Study Management Module* (SyMM) is a book-keeping module which keeps track of the student's learning session. The following knowledge is contained in the SyMM:

- Point of study suspension: time, day
- Current topic
- Current learning path: which nodes have been traversed, and order of traversals
- Which tutor questions were answered correctly, and length of time it took
- Which tutor questions were not answered correctly
- Number of instructor referrals, and at which topics or nodes
- Points of remedial branches
- Notion of understanding of current concepts

These records are used by the tutor to keep track of what the student is learning and at which points to resume after remedial branches or after study breaks. The Student Modeling Module also uses the information in the SyMM to model the student's understanding. The need for the SyMM cannot be over-emphasized, since it constitutes the basis for individualization of instruction. It is important for the system to keep track of how the student is performing at each point, to allow for adjustment of tutoring techniques to suit the student's own pedagogical needs. Moreover, the human instructor needs to refer to these records in order to know where the student needs further assistance, particularly since the computerized tutor will give the student instructor-referrals from time to time.

The *Student-Modeling Module* (or Modeler) employs the information maintained by the Study Management Module to compute a formal representation of the system's perceived level of student understanding. Several student modeling techniques have been documented. For example, Baril et al. (1991) described their student modeling using confluences. This involves the calculation of differences (deltas) between an initial state and the current state of student knowledge and the final comparison between

the student's computations and the resident expert's. This technique is however restricted to the domain of the physical sciences, where the differences between states can be explicitly computed. Moreover, the student model is finally compared (or overlaid) with the resident expert model. Most student modeling techniques can be reduced to the overlay method described in Chapter 3. We shall not use an overlay model but a strategy-independent model.

The two basic evaluation strategies (overlay and buggy) models have been criticized as being too restrictive and too intrusive, respectively. The overlay model does not allow for flexibility in student response format, since a student is expected to respond exactly the same way the resident expert does. Moreover, intermediate steps to a solution are not usually considered, thus making the tutor lose a great deal of the student's thinking process in the solution of a problem. On the other hand, the buggy model uses a set of mal-rules as a means of determining the student's deviations from correct knowledge, performs immediate evaluation of each step, and does not allow the student to work independently. The buggy model still steers the student in a particular expert-oriented direction (only it considers intermediate steps), and does not give the tutor enough time to observe the student's unique idiosyncrasies (Nwana, 1993).

Our design alleviates problems inherent to the overlay and buggy models by buffering the student's work completely before the evaluation process commences. The tutor therefore gets a global view of the student's solution (as opposed to the usual final-answer overlay model) and allows the student to solve problems independently, thus permitting the exhibition of the student's styles and otherwise hidden misconceptions which may not have been observed with the intrusive buggy model. We do not employ a mal-rule set because it is difficult to represent completely all the student misconceptions that could arise from a piece of correct knowledge.

We shall avoid the need for more complex computations by using the information provided by the Study Management Module to estimate the student's level of understanding. More detailed discussion of aspects of the Student Modeling Module will be presented in Chapters 5 and 6, but we summarize them as follows:

- traces the knowledge base to obtain alternative reasoning strategies apart from that of the Expert Problem Solver
- analyzes the student's inputs and traces the knowledge-base to find areas of the student's deviation from correct knowledge
- provides a model or system-notion of the student's beliefs based on results of the tracing processes above
- performs appropriate belief revision to represent the student's beliefs
- informs the Tutor of the appropriate tutoring complexity level for the student
- advises the Tutor whether to proceed to more advanced material or more fundamental material, based on cumulative information of the student's learning trend

The *Tutoring Strategy Module* (Tutor) is a procedural unit which contains the fundamental algorithms governing the student-system interaction. Our design allows the Tutor to conduct two learning modes: plain learning, and problem-solving. The plain learning mode involves the student just studying material contained in nodes, for the purpose of acquiring scientific or fundamental knowledge of concepts. In the problem-solving mode, knowledge acquired during plain learning is applied to the solution of problems in the domain. Most problems encountered during the problem-solving mode are case studies or real-life events in which the student may be found after learning. This is therefore similar to simulations, and encourages the student to apply knowledge without the fear of hurting lives or causing far-reaching damages.

Our problem-solving follows the *differential diagnosis* approach which we will define formally in Chapter 5, but since we are presently describing the tutor, it is necessary that we examine reports about diagnosis which are considered in the design of our tutor's interaction with a student. Arocha et al. (1993) have investigated diagnostic styles in learners and have deduced that novice and advanced learners differ in diagnostic

approaches. Novice learners employ a backward reasoning (goal-driven), depth-first search technique, which works as follows:

a) generate a single hypothesis, H_i
b) search (or query the system) for evidence to prove H_i
c) backtrack to an alternative hypothesis H_j, if H_i is proven false

This backtracking seeks to produce a hypothesis H_j which conforms to the updated data. Advanced learners (and experts in general) however employ a forward-reasoning (data-driven), breadth-first approach, which works as follows:

a) generate a set of (differential) hypotheses which may account for currently available data or symptoms
b) allocate levels of likelihood (or confidence factors) to each hypothesis
c) query the system to obtain more relevant data
d) use incoming data to confirm or disconfirm individual hypotheses in the set generated in step a). More hypotheses may be added based on new facts. However, we may require, for simplicity, that the initial set of hypotheses be complete in itself, and only eliminations be performed in subsequent refinement. Disconfirmed hypotheses are eliminated
e) repeat steps c) and d) until a single hypothesis is obtained, which matches all the available data

Artificial intelligence research (Rich & Knight, 1991) shows that goal-driven or backward reasoning is more time-consuming on the average and is employed when there is inadequate knowledge of the domain. Forward-reasoning on the other hand, is faster, more efficient, and reflects advanced domain knowledge of the user. This was confirmed by Arocha et al. (1993).

To accommodate both novice and advanced learners, our tutor is designed with the capability to present problems at both levels. The two problem-solving levels are as follows:

Novice level

a) the DQM presents a problem and includes all necessary data (complete knowledge)
b) the student and Expert-Problem-Solver are expected to each return a single hypothesis
c) the Tutor performs coaching if the student's hypothesis is incorrect.

Note that the EPS's single hypothesis is assumed to be always correct, since the complete information guides the knowledge base to present an accurate diagnosis.

Advanced level

a) the DQM presents a problem, with incomplete information
b) the student and EPS each returns a set of hypotheses
c) the student is required to order (ask the system for) desired laboratory results or other relevant data
d) the student and EPS return updated hypotheses, based on currently updated data
e) steps c) and d) are repeated until the student reaches a final hypothesis
f) the Tutor performs coaching if the student's hypothesis is incorrect

At the beginning of the first problem-solving phase, the system assumes that the student is at the novice level. Thus all relevant information is provided for the student's diagnostic activities at this stage. Based on the student's performance (speed of responses and percentage of correct diagnoses), the DQM gradually increases the level of problem complexity. This is characterized by the gradual reduction of the amount of information provided for the student. There is therefore a smooth, dynamic and system-effected transition from novice learning to advanced learning. For instance, at the first advanced stage, the DQM may remove one piece of relevant data. When the student has shown evidence of adeptness at that level, the DQM removes two pieces of data, and so on. At the advanced problem-solving level, the student is expected to ask the system relevant questions which would help to fill in the missing blanks. The number of questions posed before the student obtains the missing data, is a measure of the student's mastery of the problem-solving techniques. The following pseudo-algorithm is applied during the Tutor's interaction with the student.

```
Ask for learning mode
If plain learning then
   begin
    ask whether to start all over or from point of last stop
    check for pre-requisite knowledge
    (ask questions from preceding material or pre-requisites)
    bring up last referral
    ask questions
    if response not correct, then replace referral
    For current concept begin
            display browser overview
```

```
            receive student's choice of next desired node
            while node choice <> "" do
                    perform node-present
                    ask if to continue or suspend
                    if to continue then receive next node
                    else store current statistics in SyMM
            end
    end
Else (i.e. problem-solving)
   begin
    while not suspend do
            present case-history
            present cues
            ask for student's hypothesis, R
            if R correct then perform correct-R
            else perform incorrect-R
            ask if to continue or suspend
            if to continue then loop
            else store current statistics in SyMM
    end
   end
procedure node-present
    begin
            present material
            ask question
            receive student's response, R
            if R correct then perform correct-R
            else perform incorrect-R
    end
procedure correct-R
    begin
            give encouraging comment
            ask how student arrived at response
            if reason correct then return
            else find contradiction
            explain contradiction
            ask follow-up question
            if response correct, perform correct-R
            else perform incorrect-R
    end
procedure  incorrect-R
    begin
            ask how student arrived at response
            find and present contradiction
            give hint(s)
            repeat question
            If response okay return
            else present remedial stuff and perform tutoring process
            repeat question
            if response correct return
            else begin
                    refer to human instructor
                    give related references
                    increment no of referrals
                end
    end
```

A few comments will be timely at this point. A student indicates which node to start with from the hypermedia browser presented by the Tutor. When a question is asked

and the student's response is correct, it is important to query for the student's reasoning, because a response may be right while the underlying reasoning may be incorrect. The process of handling a student's incorrect reasoning may involve several deviations or branches from the main concept under discussion. The SyMM keeps track of these using a stack data structure, which helps the system to know where to return after such branches. On resumption of a study, it is important for the system to find out whether previous referrals have been tackled. It is also possible for a concept to become clearer after the student has studied nodes following the point of referral. The system constantly needs to update the referrals data structure, to delete concepts which have currently been understood by the student.

A student would normally have questions for the Tutor. Explanations of the Tutor's reasoning are obtained from the links in the Knowledge base. Such links are used to retrieve relevant nodes from the hypermedia component's electronic textbook, in order to further explain concepts to the student.

The Tutor could also include voice output. Fletcher (1992) and Rickel (1989) agree that people recall 25% of what they only see or only hear, 45% of what they both see and hear, and 70% of what they see, hear, and do. This agrees with the traditional classroom environment in which students grasp concepts faster if the instructor talks about it, than if they only read from a textbook. We therefore recommend a voice output mechanism, which audibly speaks every word from the Tutor, for learning enhancement purposes.

Another important issue to consider is that of tutor intervention during the student's problem-solving activity. *Intervention* implies that the Tutor interferes with the student's solution process, does not allow the student to complete his/her current state of activity, and may steer the student in a direction that is completely different from his

current train of thoughts. The timeliness of tutor intervention is a very important research issue in intelligent tutoring, and has been addressed by Nwana (1991, 1993).

The Tutor intervention spectrum ranges from the *discovery* style in which the student is not interrupted by the tutor, to the *model-tracing* style proposed by Anderson (1984), in which the Tutor closely monitors the student's activities and interrupts whenever the student makes a mistake or steers away from a prescribed thought pattern. The discovery style has the disadvantage that a student may make several mistakes without knowing it, since there is no input from the Tutor. Model tracing, on the other hand, is too intrusive and distractive, and does not clearly model the student's individual style, since it does not allow the student to solve the problem at hand at a personal pace.

Anywhere within this spectrum, the Tutor can interrupt student activity, but the question is: When? Late intervention implies that the student is left to flounder helplessly, while early intervention may mean that the student is annoyed by unwelcome intervention from the Tutor because the student feels that he could have 'got there' anyway, if the Tutor had not interrupted.

To solve this intervention timing problem , out tutor is designed to take what we call the *Collection and Invitation* approach which works as follows:

a) Collection: During the student's normal problem-solving activity, the Tutor would not intervene, but would merely collect and buffer the student's solution, and allow the student to solve the problem his own way. The Tutor's evaluation of, or comments on, the solution, would therefore not commence until the student has completed the solution process to his/her own satisfaction. It is called a *normal* problem-solving activity because the student actually completes his/her solution.

b) Invitation: An "abnormal" problem-solving process is one in which the student cannot complete a solution but reaches an impassé and flounders at some point during the solution process. At this point the student would issue an explicit *invitation* to the Tutor; that is, the student requests the Tutor's assistance.

In both cases, the Tutor must be unobtrusive and not annoy or distract the student with unwelcome interruptions. The "Collection and Invitation" intervention technique has the added advantage that the Tutor can "replay" the student's solution for optional review (process during which the student may modify responses if s/he wishes) before evaluation actually commences. The "Collection and Invitation" technique, as well as the added feature of solution-review, are unique to this design. Some systems like GUIDON (Clancey, 1987) do have a HELP facility whereby the student explicitly requests the Tutor's help, but they are not strictly "Invitation" systems because the Tutor sometimes interrupts the student's problem-solving activity without invitation.

The functions of the Tutoring Strategy Module (Tutor) are summarized as follows:

- dynamic adaptation to the student
- uses browser overview to identify nodes to be taught
- pre-tests the student to ascertain knowledge of pre-requisites
- retrieves appropriate nodes from the electronic textbook
- speaks audibly to the student
- assumes a middle ground of tutoring complexity, at the start of learning
- informs the Dynamic Questions Module (DQM) of desired complexity level
- provides DQM with keywords or data to fit into schema
- receives questions from DQM
- evaluates the student's response
- keeps track of student's response speed, as a measure of understanding
- moves up or down the complexity scale if the student's speed and accuracy become higher or lower, respectively
- branches to obtain remedial material if needed
- adopts a "collection and invitation" policy by buffering the student's solution and not interrupting the student's problem-solving activity unless explicitly requested to do so
- queries the student for reasons behind student-responses
- automatically chooses the next node to be taught, or allows the student choice in alternative cases
- allows the student to navigate forwards or backwards, to revise, suspend learning or to quit from the course altogether

- informs the Study Management Module (SyMM) of all visited nodes
- saves points of branching for retrieval on return. Stores these data into SyMM if learning is to be suspended
- discontinues with the current concepts into more advanced or more fundamental material, based on the student's performance
- administers tests from the DQM, but with an option for response-review, as in the pen and paper environment
- provides dual-mode tutoring: straight learning and problem-solving
- provides a dynamic and smooth transition to/from novice learning to/from advanced learning, as the student's mastery or confidence level changes
- attaches appropriate object procedures or methods to icons in the electronic textbook. These methods are fired when the appropriate icons are selected
- obtains correct problem solutions from the Expert Problem-Solver (EPS), which solutions are used as guides in pedagogy and evaluation of student responses
- allows the student to ask questions
- obtains explanations from the Expert Problem-Solver
- communicates directly with all other components of the system, and thus constitutes the very heart of this project

More aspects of the tutoring strategy module design are discussed in Joshua & Scuse (1995).

4.4 Expert system and hypermedia synergy

The question arises as to why an expert system and a hypermedia system should be merged in an ICAI application.

Granted, most existent ICAI systems are built around the expert system paradigm. However, First et al. (1985) admit that much of the knowledge bases have been in textual form. Even in some attempted graphical applications, they have been mostly still diagrams or scanned stills. Since the expert system technology has intelligence but cannot effectively store advanced graphics, especially those involving complex animation, an augmenting component is needed which would allow such systems to have the intelligence of an expert system as well as the ability to display text or graphics as desired by the user. This is particularly pertinent in simulations or other real-life applications like medicine, the military, or nuclear reactor environments. The

hypermedia system therefore fits into this structure and will provide the interface needed for the storage and retrieval of a variety of learning material structures.

Secondly, the hypertext paradigm provides essentially a storage, retrieval, and browsing environment, and consequently lacks intelligence and computational abilities. This is probably the reason these systems have not been widely applied to ICAI, which requires intelligent manipulation of stored nodes. Halasz (1988) and Rada (1991) emphasize the need for an inference engine, for easier and more intelligent manipulation of hypermedia systems. Kurland et al. (1992) claim that their system is a hypermedia learning system, but it actually functions in two modes: as an expert-system-based simulation learning system and as a hypermedia-based browsing system; in the hypermedia mode, the system does not intelligently tutor the student. Akscyn et al. (1988) have also done some tutoring with KMS, but admit that it has only been minimal since KMS is fundamentally an authoring hypermedia system. Owing therefore to the inherent weakness of the hypermedia paradigm (lack of deductive ability), an intelligent interface is needed which will enable ICAI applications to possess the advantage of hypermedia environment as well as the ability to tutor a student intelligently and perform accompanying computations. Some hypermedia systems are accompanied by programming components. An example is HyperTalk, the programming subset of HyperCard. These components are however restricted in the sense that they encapsulate only hypermedia-related functions like link procedures or movement of nodes as objects within the system. They are therefore not adept at programming with respect to other (non-hypermedia) functions. A more versatile programming facility is therefore needed which can manipulate both hypermedia and non-hypermedia constructs. The expert system technology meets this need. Compared to hypermedia systems, an expert system allows for more flexible interaction with a student. Since a student can use rules to directly access specific nodes, the sometimes frustrating navigation through irrelevant

nodes is avoided. Also, an expert system can derive new data from existing ones, using its computational facilities.

Jao and Hier (1993) have combined the expert system and hypermedia technologies in their Intelligent Stroke Consultation system. This system however differs from our model, since the ISC is merely an information retrieval system.

4.5 General model summary

In summary, our design model is as described above. The following facilities are provided by the model:

- provision of focused guidance to student
- allowance for individualized learning paths as needed by student
- maintenance of an efficient student model
- maintenance of an efficient expert model
- generation of intelligent explanations and help
- generation of dynamic questions
- provision of extensive graphics features using a hypermedia system

Implementation could be done on any type of computer system, but we choose a microcomputer environment because the future trend of computing points to home and personal computing, along with the use of microcomputers in virtually all spheres except for large businesses. Moreover, most CAI students can have easy access to microcomputers since they are cheaper than the industry-directed minicomputers and mainframes.

We hope that this design will go a long way to removing some of the individualized instruction hindrances facing tutors in our time, particularly the problem of modeling the student's understanding, and adapting instruction in response to student needs. We also hope that our work will open the door for more extensive research on the marriage between hypermedia and expert systems technologies in Computer-Assisted Instruction and other application areas.

4.6 Restricted Focus

Considering the time-constraints imposed on a single Ph.D. research, it is impossible to implement all the components of this model. We shall dedicate the rest of this thesis to issues surrounding the Modeler, which constitutes the heart of intelligent tutoring systems, and still present open problems particularly in the areas of identifying student misconceptions and maintenance of student beliefs. In the next chapter we will discuss different parts of our student modeling module, along with a review of research that has been done in belief revision. We will then present a new belief revision scheme in Chapter 6 along with algorithms which can be used to implement components of the Modeler. Although the tutor and the student modeler are closely linked as we pointed out earlier, the reason we have chosen to focus on the modeler rather than the tutor is because the tutor cannot perform its functions effectively without accurate input from the modeler. It is therefore important to have a functional modeler before implementation of the Tutor can be considered.

CHAPTER FIVE: FOCUS ON THE STUDENT MODELING MODULE

In this chapter, we focus on the central component of our research: the Student Modeling Module (or Student Modeler). We first describe the overall Student Modeler model, emphasizing the justification of our unique approach which deviates from the commonly used overlay and buggy or mal-rule approaches (algorithms will be addressed in Chapter 6). Since student modeling is essentially a belief centered activity, we present a detailed discussion of belief maintenance and belief revision issues which will be considered in our own model.

5.1 The Student Modeling Module

As shown by previous discussions, the Student Modeler is a salient aspect of intelligent tutoring. Human tutors can perform modeling well because they are able to observe the student's facial expressions, vocal inflections, a slow-down or speed-up in speech, nervous hand-motions, or other physical signs which suggest the student's level of comprehension of a concept under discussion. On the other hand, a computer tutor has only the student's typed responses and a measure of response speed available for its modeling task. It is therefore more difficult to model the student's thinking, since the physical features cannot be "seen" by the computer.

The student modeling problem has been described as an *intractable* task which may be seen by some as practically impossible (Self, 1988). However, Self suggests that if we streamline our goals to more practicable levels, we can obtain some useful student-modeling in our intelligent tutoring systems, in spite of the limited information available to them (Self, 1988). The emphasis in the design of our Student Modeler was on using information that is easily obtained in order to build a more sophisticated modeling component.

The most commonly used approaches to student modeling are the *overlay* and *buggy* (or *mal-rule*) methods. As discussed in Chapter 4, the overlay model has been criticized as being too restrictive and does not correctly model expert behavior since experts use different approaches to solve problems. It does not motivate a student since it does not allow for the student's own approaches if they differ from some previously programmed approach. It therefore hinders individualization which is a major objective of intelligent tutoring. We must however state here that some domains naturally lend themselves to overlay modeling. A typical example is the live electrical appliance maintenance of the EEMT system, and the industrial boiler of the RBT system described in Sections 3.3. Other domains include operations of medical equipment, nuclear reactors, power plants (and in fact, the operation of most physical equipment) which may be life-threatening or hazardous and highly procedural. However, perceptive, social, decision-making, and managerial domains do not lend themselves to the overlay model, because of the multiple strategies involved in these domains. Our prototype domain is a perceptive, diagnostic domain and does not lend itself to the overlay technique, since psychiatrists do not usually follow the same thought path to diagnose a mental disorder.

On the other hand, the buggy technique has been criticized particularly because of the use of mal-rules which are usually not exhaustive and do not represent idea dependencies and inter-relationships between misconceptions. It is also computationally expensive to search through a set of mal-rules simply to identify a bug (which may not even be represented in the set) when the system can rather identify the correct piece of knowledge which has been mis-understood, and then discuss with the student to determine the exact nature of the misconception.

5.1.1 A knowledge-directed, strategy-independent approach

We have taken a different approach to student modeling: a knowledge-direct, strategy-independent technique. This means that:

a) The student is not restricted to a particular problem-solving strategy or path. Student styles and idiosyncrasies are allowed, as long as the content of the student's response represents a correct solution to the problem at hand.

b) We do not use a mal-rule set, but compare the student's misconceptions to correct knowledge in the knowledge-base.

The implications of this are:

i) The design models the student's problem-solving activity (a student solution history), and for effectiveness, the student Modeler may get the Tutor to advise the student if some aspects of his/her problem-solving approach are sub-optimal. "Sub-optimal" here does not imply the presence of a single or target optimal behavior, but the system (through the Expert Problem Solver) will be able to advise the student on things like problem decomposition, or collection of multiple steps into one, and so on.

ii) Since a mal-rule set is not used, it is left to this model to discover the exact cause of the student's misconceptions. This is a large improvement over the buggy approach because our model can therefore identify the inter-connections of misconceptions, wrongly used analogies, et cetera, that have been applied by the student.

5.1.2 Prototype domain

Before we discuss the student modeling aspect of our work in greater detail, it is important to introduce the environment in which we will apply the Modeler itself. Our prototype domain is psychiatry, with particular emphasis on the differential diagnosis of psychotic disorders using the guidelines laid out in the American Psychiatric

Association's DSM-III-R manual (The American Psychiatric Association 1987; Kaplan & Sadock 1985). Differential diagnosis may be defined as the recursive construction and refinement of a set of hypotheses, using available data to eliminate or disconfirm certain hypotheses until a set remains which explains all the available data. This is the same approach used by Moreno & Plant (1993) to assist clinicians in the proper diagnosis of mental disorders. However our work differs substantially from Moreno & Plant's since theirs is similar to the role of our tutoring model's Expert Problem Solver which dynamically updates a current differential diagnosis based on evidence or symptoms, while ours involves diagnosis as well as tutoring and the complex task of student modeling.

5.1.3 Modeler components

As shown in Figure 5.1, the Student Modeler is designed to perform its task in several phases.

1) Receipt of student's response from the Tutor. This is a trivial sub-task performed by the *Receiver*.

2) Query analysis: Query analysis is the stage in which the modeler would break down the Tutor's question to determine its exact contents and structure. It is performed by the *Query Analyzer*. The aim of this stage is to identify the key elements required by the Tutor. For example, if the Tutor asks the question:

What is the diagnosis if incoherence = T and duration ≤ 4 ?

the Query Analyzer would identify the query portions as follows:

Given:	Duration ≤ 4
	Incoherence = T
Required:	Diagnosis

This analysis facilitates a more directed search in the next phase of student modeling.

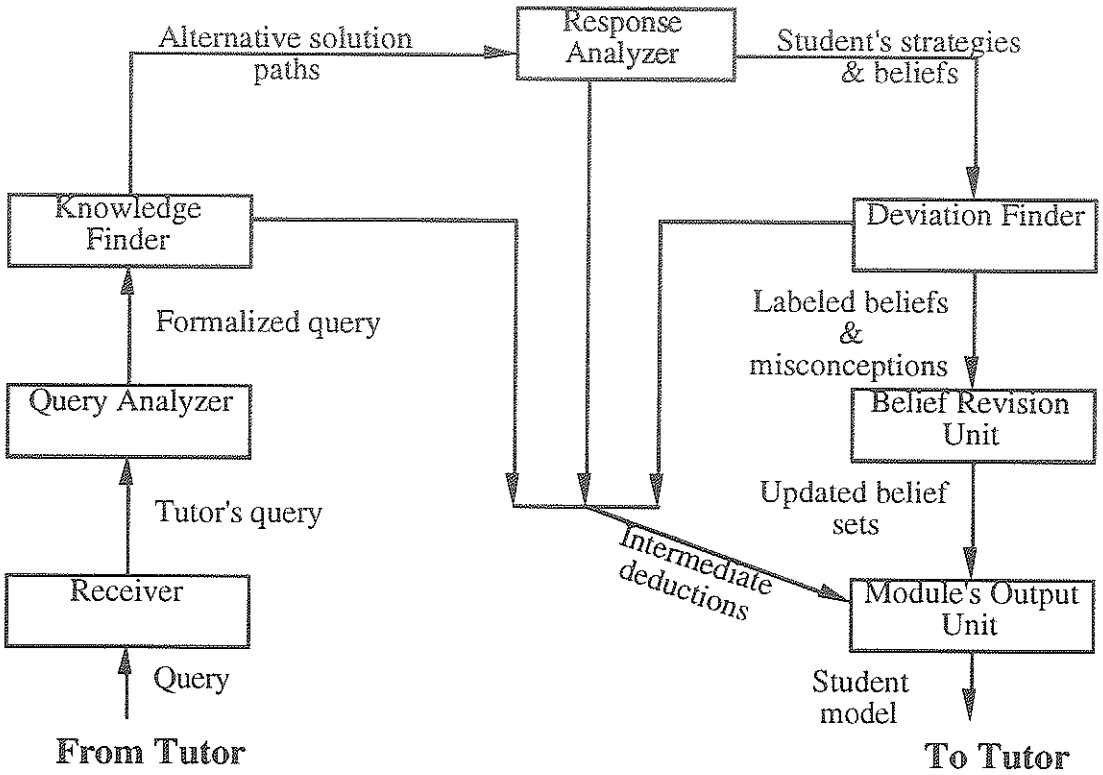


Figure 5.1 Student modeling phases

3) Location of alternative solution paths: This is performed by the *Knowledge Finder* which will obtain its search leads from the Expert Problem Solver (EPS). Using the solution to a problem obtained by the EPS, the Knowledge Finder processes the knowledge base to identify alternative paths to the same solution. This is a more detailed search than the single path discovered by the Expert Problem Solver, as we show in Chapter 6. The identification of these alternative paths will enable the Modeler to perform solution-strategy-independent modeling, since the student would be allowed to use any of the discovered paths to obtain a correct solution.

4) Response Analysis: This is similar to Query analysis, but this time the object is the student's response. The *Response Analyzer* will take the student through queries aimed at obtaining the student's beliefs about a problem being solved. Using these beliefs, the

Response Analyzer would then examine the alternative paths computed by the Knowledge Finder in stage (3) above, in order to identify what reasoning path(s) the student might be applying in his/her solution. Candidate paths selected by the Response Analyzer will help the Modeler to obtain a better model of the student's style and thought patterns.

5) Location of deviations: This is performed by the *Deviation Finder* which will process the knowledge base and determine whether each belief identified in (4) is correct or incorrect. For incorrect beliefs, the Deviation Finder looks for knowledge in the knowledge base which represent some forms of the student's deviations, so as to explain the student's possible misconceptions. The Deviation Finder also identifies missing or incomplete information in terms of beliefs which the student should have had for a complete solution to the problem at hand, but which are missing from the set of beliefs collected by the Response Analyzer in stage (4) above.

6) Belief Revision: This is the update process in which the *Belief Revision Unit* will modify the current student belief base to include newly discovered misconceptions and correct beliefs. Some beliefs are derived from other beliefs, and the Belief Revision Unit must determine what to do with different beliefs in the event that a new contradictory belief is observed from student input.

7) Return belief information to the Tutor.

Algorithms for each of the above phases are described in Chapter 6. However, in the rest of this chapter we shall discuss Phase 6 in more detail, since it is a very salient part of the Student Modeler.

5.2 Belief maintenance and belief revision

There is no doubt that student modeling in intelligent tutoring is an artificial simulation of the human teacher's attempts to understand the student's beliefs about concepts. In artificial intelligence terms, a belief system comprises a set of beliefs along with procedures for acquiring new beliefs (Rapaport 1992). We will extend this definition to include procedures for maintaining consistency in the belief system, which extension is in the area of belief revision and will be discussed in detail later in this chapter. Beliefs must not be confused with knowledge, since Rapaport has defined *knowledge* as "justified true belief" and belief systems have wider scopes than knowledge systems. This definition leads to a conclusion about beliefs which constitutes a major focus of our Modeler's Deviation Finder: a belief may not necessarily be a fact. One can believe false propositions or nonexistent objects, while objects of knowledge are always true and existent. It is therefore the Modeler's task to identify which of the student's beliefs are true and which are false. A typical example of false beliefs is Colby's (1975) simulated paranoid (PARRY) who believes that everyone (especially the consulting psychiatrist) is out to destroy, humiliate, or insult him.

5.2.1 Properties of beliefs

Beliefs have several properties as discussed in Abelson (1979) and Cohen & Feigenbaun (1982). We shall relate some of these characteristics with intelligent tutoring presently.

Beliefs are *non-consensual* in that people hold different views about the same object of belief, explaining the wider scope of belief over knowledge as mentioned above. A tutoring example is a case of two students understanding the same concept differently. Beliefs also deal with conceptual entities (for example, topics) and since beliefs differ across agents (holders of beliefs, or believers), an entity in one agent's belief system

may be absent from another agent's belief system. This is why a student has a separate belief base which differs from a tutor's belief base (or Knowledge-base). The student's belief base contains beliefs which the Student Modeler has observed or deduced from student inputs, and some of which may conflict with the tutor's beliefs represented in the Knowledge-base.

Beliefs have affective, emotional, or confidence components. One agent may feel very strongly about an object while another may be nonchalant. In the application area of tutoring, affective components may be represented using certainty factors or percentages, and a student may be required to suggest possible causes of a disorder ranked in order of perceived responsibility. Beliefs are also related to an agent's environment, circumstances, and past experience. In tutoring these may be linked to pre-requisite, contextual, and heuristic knowledge.

A belief space is defined as a set of beliefs along with the set of items about which the beliefs are held (Rapaport 1986). In tutoring, the set of items about which beliefs are held are the concepts discussed during learning, which is the same for both a tutor and a student. The two agents' belief spaces however differ because the sets of beliefs are different.

5.2.2 Belief representation issues

5.2.2.1 Issues

Before we discuss approaches which have been employed in the representation of beliefs, it is important to address some issues and problems surrounding the representation of beliefs. These issues arise from the nature of beliefs themselves, the interpretation of beliefs in a multi-agent environment, and particularly from the basic fact that beliefs differ from knowledge. These properties, which make belief systems more difficult to maintain than pure knowledge systems, have been defined in the foregoing

section. Essentially, as discussed in Rapaport (1986, 1992), the major problems faced in the representation of beliefs are as follows:

- representation of coreferential opacity (de dicto) versus coreferential transparency (de re)
- representation of an agent's view of his/her own beliefs (self-reference) versus representation of a system's view of the agent's beliefs
- representation of third party beliefs
- representation of pronominal reference (anaphoric, indexical, or quasi-indexical)
- representation of emotional or affective components

These issues are relevant to tutoring, especially the first and the last; however for the purpose of review completeness we shall discuss these representations presently.

Coreferentiality refers to a case of the equivalence of more than one object, and an agent's belief about all the different occurrences of that object. For instance, if "Ms. Katie Williams" is "The Human Resources Manager" as well as "The Owner of the Manitoba Angels", and it is known to a belief system's control knowledge-base that "Ms. Katie Williams is away at Norway for holidays", the system would know that this last statement is equivalent to "The Human Resources Manager is away at Norway for holidays" and "The Owner of the Manitoba Angels is away at Norway for holidays". However suppose that an agent A believes that "Ms. Katie Williams" is "The Owner of the Manitoba Angels", but does not believe that "Ms. Katie Williams" is the "Human Resources Manager", then coreferentially speaking, A believes *de re* (referential transparency) that "The Owner of the Manitoba Angels is away at Norway for holidays" and believes *de dicto* (referential opacity) that "The Human Resources Manager is away at Norway for holidays".

Simply put, belief *de re* refers to an agent's belief in something that is true while belief *de dicto* is an agent's lack of belief (or ignorance) about something that is true. In both cases, the object is true (a piece of knowledge), but the problem is how to represent an agent's belief or ignorance about it. Although Rapaport identifies the need for proper

representation of these differences, he does not suggest how the representation should be done. However, the contemporary practice has been for a system to assume that the agent A believes all coreferential facts about an object, until A shows evidence to the contrary (Doyle 1979, de Kleer 1986; Huang et al. 1991). This is discussed further under the deduction of new beliefs.

Self-reference refers to an agent's characterization of his/her own beliefs. For instance, if Mary says to herself, "I am beautiful", it would be incorrect for a belief system to report that belief as "Mary believes that I am beautiful". Rather the system would represent the belief as "Mary believes that she is beautiful", or "Mary believes that Mary is beautiful". This also applies to third party reports (or deeper levels). For instance, if John heard Mary compliment herself, it could be represented as "John believes that Mary believes that Mary is beautiful" and not "John believes that Mary believes that I am beautiful". The phenomenon of self-reference and third- or multi-party reference is termed *nested beliefs* by Rapaport (1986) who has used semantic networks to address the problem of interpretation, with each node representing a belief (for example, "I am beautiful") or an agent (for example, Mary), and arcs clearly labeled to convey the correct semantic interpretations. We must however note that in individualized tutoring, we are faced with a two-agent belief system in which it is easier to represent tutor and student beliefs than in multi-agent systems. Furthermore, with respect to a Student Modeler, the number of agents is implicitly reduced to one (that is, the student), since the Modeler is trying to identify the student's beliefs only. This makes the representation of student beliefs easy to represent as stand-alone beliefs without agent identification. This means that rather than "The student believes that (Manic syndrome is True) $\Rightarrow$ (Patient has delusional disorder)", the Modeler can implicitly represent the student's belief as "(Manic syndrome is True) $\Rightarrow$ (Patient has delusional disorder)", without confusion as to whose beliefs are being represented.

Pronominal reference addresses the contextual interpretation of pronouns in belief reports. Rapaport identifies three contextual uses of pronouns: *anaphoric reference*, *indexical reference*, and *quasi-indexical reference* of pronouns. Anaphoric reference describes a case where a pronoun's antecedent occurs within the text under examination. For example, in "John is tall and *he* is clever", *he* is interpreted as *John*. In indexical reference, the pronoun refers to an entity outside the text. For example, "That woman over there is called Ms. Barbara Adams. "She is a philosopher". In the context of "She is a philosopher", *she* is indexical because it refers to "That woman over there..." which is outside of the statement's immediate context. In quasi-indexical reference, a pronoun occurs within a belief context but outside of its own level of nesting. For example, "Mr. McCarthy owns this hotel. John believes that he is very rich", in which *he* does not refer to John (although it appears to belong to that context) but to a third agent (Mr. McCarthy). The problem with pronominal reference arises in a nested belief environment mainly because the pronouns are not structured to refer to explicit objects. For instance, there would have been no confusion if the last example had read "Mr. McCarthy owns this hotel. John believes that Mr. McCarthy is very rich". However, this is not the way people speak in natural language. Again Rapaport addresses the pronominal reference problem using semantic networks (Rapaport 1986). It must however be noted that in an application that does not handle natural language representation, it would be sufficient to allow agents to communicate in a structured language that avoids ambiguity. For instance in a tutoring system, objects should have explicit names and a student and a tutor must be required to use those names. For example, the student-input "Visual hallucinations suggest the presence of schizophrenia" is clearly more explicit than "They suggest the presence of schizophrenia", at which input the tutor may request further clarification.

Representation of emotional aspects of beliefs have been addressed mostly using certainty factors, as in GUIDON (Clancey 1987). Beliefs would normally have certainty factors ranging from 0 to 1. This shows a clear difference between beliefs and knowledge: knowledge objects always have a certainty factor of 1, since they are facts; programmers of belief systems however have to estimate certainty factors, which may actually not be accurate measures of an agent's emotional feelings about the belief(s) being represented.

5.2.2.2 Representation

To a large extent, reasoning about a belief system is affected by the representation of the constituent beliefs. Although beliefs differ from knowledge, they can be represented in the same ways, in a closed world fashion, with each belief system represented as a separate belief base. To represent a belief, Abelson & Reich (1969) developed the *implicational molecule* which was a list in which the first part of the molecule implied the second part. For example

[A has-feathers, A can-fly],

which can be rewritten as

(A has-feathers) $\Rightarrow$ (A can-fly)

One can apply the universal classifier to this as follows:

$\forall$ A: (A has-feathers) $\Rightarrow$ (A can-fly)

or $\forall$ A: has-feathers(A) $\Rightarrow$ can-fly(A) in predicate logic form. This can be instantiated as desired. In rule form, the same belief can be represented as

IF (A has feathers) **THEN** (A can fly)

The implicational molecule is typically used in single-agent or double-agent representation in which an agent is trying to express his own beliefs or those of one other agent. In either case, there is no confusion as to whose beliefs are being

represented, as opposed to a multi-agent environment where there must be an explicit identification of the agent being modeled in order to avoid confusion. For instance, (John believes that Mary is beautiful) may be a belief expressed by yet a third agent. In the ICAI environment, and with particular emphasis on the Student Modeler, the belief base is regarded strictly as a collection of the student's beliefs, as has been discussed earlier. Therefore every belief represented therein is known to belong to the student, which makes it unnecessary to express beliefs as

(The student believes that [IF (A has feathers) THEN (A can fly)]).

It is sufficient to represent this belief simply as

IF (A has feathers) THEN (A can fly)

Colby (1969) also employs the list structure in the representation of beliefs. For example,

(F (agent Barb) (action like) (object children))

means "It is a Fact that Barb likes children"

Lists are very easy to manipulate and can be viewed as frames with slot labels which may be absent but which are implicitly known by the programmer.

Rapaport (1986) applied semantic networks, using the Semantic Networks Processing System (SNePS), in the representation of multi-agent beliefs. Each belief is represented in a separate node and nodes are linked meaningfully. The same representation approach is also used by Maida & Shapiro (1982) and Martins & Shapiro (1988); both projects are also on nested multi-agent interactions.

At this point we must note that the techniques applied for the representation of a belief system are determined by the nature of processing done with the beliefs, particularly, the deduction of new beliefs from old beliefs, and the revision of beliefs. It is important that

these two important relations to belief-representation be discussed before we explain the elements of our belief model.

5.2.3 Deduction of new beliefs

Rapaport's definition of a belief system include " . . . procedures for acquiring new beliefs". These refer to inference rules used to derive new beliefs from older beliefs (Maida & Shapiro 1982; Martins & Shapiro 1988; Doyle 1979; Huang et al., 1991). For example, in tutoring arithmetic, a tutor may observe that a student understands the principles of subtraction and so applies inference rules to deduce that the student must have previously known the principles of addition, since addition is normally a prerequisite to subtraction. This would give us the following (comments are italicized):

a)	knows(subtraction)	*{ an observed belief}*
b)	knows(subtraction) $\Rightarrow$ knows(addition)	*{an inference rule}*
c)	$\therefore$ knows(addition)	*{a derived belief}*

This example brings us directly into the classification of beliefs in tutoring. a) is a base or *observed* belief, b) is an *inference rule* which is always the tutor's belief that can be applied to student beliefs, c) is a *derived* belief. The term *base* does not apply to observed beliefs alone. There are generally two types of base beliefs: observed beliefs and what we call *base-assumptions* (or simply assumptions) (de Kleer 1986). The reason we have used the term "base-assumptions" is that derived beliefs are also assumptions made by the tutor, only they are derived using inference rules. Base-assumptions are however not derived, but are assumed by the tutor for convenience reasons. However to avoid confusion, we shall use the classifications "derived" and "assumptions" for derived assumptions and base-assumptions, respectively. Assumptions are often used in stereotypical applications, an example of which is GRUNDY (Rich 1979), a library advisor which assumed initial stereotypical classes of beliefs based on a user's self-description. For accurate belief manipulations, record

must be kept of how derived beliefs are obtained, what Barr & Feigenbaum (1982) have described as *dependency-records* which link derived beliefs to their justifications (assumptions and inference rules used to derive them). Fikes (1975) and Hayes (1975) developed dependency records for their automated planning systems in which the effects of robot actions were represented in ADD/DELETE lists where a DELETE list included all states or actions to be discarded as a result of the latest robot action.

This leads us to the philosophical theory of *intensionality* of beliefs discussed by Maida & Shapiro (1982) and Rapaport (1986, 1992). Intensionality puts constraints on the derivation of new beliefs about coreferential objects. Coreferentiality is the philosophical term for equivalence, which we have discussed in Section 5.2.2.1. The intensionality theory is that an agent's belief in an object does not necessarily imply the agent's belief in a coreferential object. This theory is based on the fact that agents do have inconsistent beliefs. Intensionality therefore constrains a system not to make any inference unless the agent shows an explicit evidence of belief in a coreferential object as well. For example, suppose that the Evening Star is another name for the Morning Star (coreferentiality) and a student believes that the Evening Star is a planet, but that the Morning Star is not a planet. We would have the following:

(Evening_Star, planet)	{student's belief}
(Evening_Star, planet) $\Rightarrow$ (Morning_Star, planet)	{inference rule}
$\neg$ (Morning_Star, planet)	{student's belief}

In the above case, it would have been logical for the tutor to derive the belief that the student also believes (Morning_Star, planet). The student however displays a conflicting belief, a case of *referential opacity*, that is, failure of substitutability of coreferential terms in this student's intensional context.

Manipulating student beliefs would have been a simple matter if the theory of intensionality were adhered to. That way, only observed beliefs would be handled,

removing the complications introduced by assumptions and derived beliefs. However, contemporary belief systems (Huang et al., 1991; de Kleer 1986; Doyle 1979) apply nonmonotonicity to beliefs, thus assuming the presence of a belief in the absence of evidence to the contrary, followed by retraction of the assumed belief when its negation is believed. For the above example, we can conclude that the tutor actually derived (Morning_Star, planet) based on the student's observed belief that (Evening_Star, planet), but had to retract the assumed belief when the student explicitly believed ¬(Morning_Star, planet). The nonmonotonic basis for derivation of new beliefs plays a central role in contemporary belief revision procedures.

5.2.4 Belief revision

Belief revision is the modification of a belief set in the light of new, possibly conflicting, beliefs (Rapaport 1992; Martins 1992). A reasoning system must be able to adapt to the dynamics of beliefs. One of the earliest belief revision systems is the Truth Maintenance System (Doyle 1979). Another is the Assumption-based Truth Maintenance System (ATMS) which is based on assumptions and not observed beliefs (de Kleer 1986). Most contemporary belief revision systems are built on the TMS and ATMS (Barr & Feigenbaum 1982; Huang et al., 1991; Martins & Shapiro 1988; Rich 1979). Essentially, belief revision introduces a temporal component into a belief system, by the removal of older beliefs that contradict incoming beliefs. The usual approach used is *dependency-directed backtracking* (Stallman & Sussman 1977). Dependency-directed backtracking involves recording beliefs along with their dependency records, such that when a conflicting belief arrives at the system, its negation is removed as well as its (the negation's) justifications and dependent derivations. Dependency records include the set of inference rules and other beliefs from which a particular belief is derived, and facilitate detection of existing contradictory beliefs. Dependency-directed backtracking is a recursive process since a negation's

justifications may also have their own justifications, and so on. For example, in the arithmetic example given earlier, suppose the student displays a new belief, ¬know(subtraction). This would lead to the removal of the conflicting old belief, know(subtraction), leading to the removal of know(addition) whose justification has now been deleted.

The selection strategy guiding removal of conflicts or culprit beliefs affects the resultant belief set. There are two schools of thought on the selection of candidate beliefs to be removed during a revision process: coherencists and foundationalists. Foundationalists (de Kleer 1986; Doyle 1979; Huang et al. 1991; Martins & Shapiro 1988) follow the TMS and ATMS approach in which all beliefs are removed whose justifications no longer exist while coherencists (Alchourrón et al., 1985; Boutilier 1994; Boutilier & Goldszmidt 1993; Dalal 1988; Gardenförs 1984, 1992; Makinson 1985) delete only beliefs that directly negate incoming contradictions, leaving derived beliefs untouched if they still *make sense*. Coherencists argue that the foundational approach is "extremely cautious", sometimes deleting too many beliefs including those which may be unrelated to incoming contradictions, and also those beliefs which still make sense even though they no longer have valid justifications. This argument agrees with cognitive research findings about the ability of human memory to recall concepts engraved into long-term memory, particularly in the face of related new information (Landauer & Freedman 1968; Underwood & Ekstrand 1968; Schustack & Anderson 1979; Anderson 1980). In fact, Anderson (1980) reveals that even if prior knowledge is momentarily *forgotten*, it is not truly lost, but can be revitalized using appropriate cues. Also, in his comparison of foundations and coherence theories, Doyle (1992), a leading foundations researcher, admits that

> "Experiments have shown that humans only rarely remember the reasons for their beliefs, and that they often retain beliefs even when their original evidential basis is completely destroyed. . . .

> Gardenförs correctly observes that the foundations approach . . . corresponds poorly to observed human behavior."

Cognitive research findings are particularly true in the domain of intelligent tutoring, in which a student's misconception of a high-level concept does not necessarily imply that the student no longer knows the underlying prerequisites or lower-level concepts. Clancey (1987) employed this reasoning in GUIDON's USE-HISTORY, USED? and SAPPLIED? student modeling parameters (refer to Chapter 3), where USE-HISTORY indicates whether the student has knowledge of a concept, USED? measures a student's use of the concept in a problem situation and SAPPLIED? measures the student's ability to apply the concept to a related or higher-level concept. In other words, a student may use a concept in a problem, but may not be able to apply the same concept to a related superconcept. We can therefore conclude that the student's inability to apply a subconcept to a superconcept does not necessarily mean that the student no longer knows the subconcept.

Representative of the foundations approach is the work done by Huang et al. (1991) in which they address the issue of *minimality* by applying a *hitting set* tree-pruning strategy in the selection of candidate beliefs in their EBRS. The hitting set H for a family of sets, F, contains at least one element of each set in F while a hitting set tree HS-tree for the same family is an edge-labeled and vertex-labeled tree whose root is labeled by a set in F (or Δ if F is empty), and in which the hitting set for each vertex v_i is the set of edge labels from the root to v_i. Essentially the tree is used to select the minimal set of candidate beliefs which represent a conflict to a belief base. The tree is subjected to a pruning algorithm at the end of which every vertex v_K labeled Δ would represent a hitting set, or a candidate (Huang et al. 1991). This candidate set includes contradictory beliefs, their dependent derived beliefs, and sometimes unrelated beliefs, as we shall show in a later example.

5.2.4.1 The AGM postulates

Much of the work in the coherencist school stems from what have become known as the *AGM postulates* (due to Alchourrón, Gardenförs and Makinson) discussed in Alchourrón, Gardenförs and Makinson (1985), Gardenförs (1992) and Makinson (1985). These postulates are logical rules used to assess the completeness and consistency of a revised knowledge base, as demonstrated by Dalal (1988) and Boutilier & Goldszmidt (1993). The postulates ensure that the set resulting from revision of a belief base with a new belief includes all of that belief's consequences or derivations, and that the revision does not introduce any new contradictions or inconsistencies to the belief base.

In developing the postulates, Alchourrón et al. defined a *consequence* operator, Cn, which is the result of transforming a set of propositions X by the application of logical operators, such that $X \subseteq Cn(X)$, $Cn(X) = Cn(Cn(X))$, and $Cn(X) \subseteq Cn(Y)$ for any $X \subseteq Y$. A more explicit definition of the consequence operator is given by Gardenförs (1992): If X is a set of beliefs and X logically entails or implies ϕ, this is written as $X \vdash \phi$. Cn(X) is then defined as the set of all logical consequences of X, that is $Cn(X) = \{\phi: X \vdash \phi\}$. Put in simpler terms, the consequence of a belief set is the set of beliefs contained in the belief set along with the beliefs derivable from them. Where derivable beliefs are stored in the same belief set as their justifying beliefs, the consequence of the set reduces to the set itself. Alchourrón et al. also define a *theory* as a set of propositions (or a belief set) X that is closed or consistent under its consequence Cn.

We now describe the postulates as follows (where $\dot{+}$ and Cn and $\dot{-}$ are revision, consequence and contraction operators, respectively *Contraction* refers to the removal of contradictions from a set of beliefs.):

(P1) $A \overset{\circ}{+} x$ is always a theory

Essentially, postulate P1 implies that a closed or consistent set of propositions A revised with an incoming proposition x, is also closed or consistent. This revision would naturally involve prior removal of inconsistencies. For example, given a belief set {a,b}, revision with an incoming noncontradictory belief x would result in {a, b, x} which is also a belief set. Revision with an incoming contradictory belief $\neg a$ would result in $\{\neg a, b\}$ after removal of the older contradiction, a.

(P2) $x \in A \overset{\circ}{+} x$

P2 implies that x is included in A revised with x. Using our previous example, each incoming belief is included in the belief set resulting from the revision process. For example, for revision with x, $x \in \{a, b, x\}$ and for revision with $\neg a$, $\neg a \in \{\neg a, b\}$.

(P3) If $\neg x \notin Cn(A)$, then $A \overset{\circ}{+} x = Cn(A \cup \{x\})$

P3 implies that revision of A with a non-contradictory x is done by simply adding x to A, as was demonstrated in the example for P1.

(P4) If $\neg x \notin Cn(\varnothing)$, then $A \overset{\circ}{+} x$ is consistent under Cn

P4 implies that the empty set is not included in elements that might cause inconsistencies in A. Putting it another way, revision with a non-empty belief will never result in a null set. For example, if A = {a, b} and b is derived from a, revision with $\neg a$ would result in the deletion of a as well as b, but the resulting set is $\{\neg a\}$ which is not equal to the null set $\varnothing$. The only situation when $\varnothing$ will result is when $\varnothing$ is revised with $\varnothing$, which is essentially a non-revision process.

(P5) If $Cn(x) = Cn(y)$, then $A \overset{\circ}{+} x = A \overset{\circ}{+} y$

P5 implies that revision with two identical sets yields two identical results.

(P6) $(A \overset{\circ}{+} x) \cap A = A \overset{\circ}{-} \neg x$, whenever A is a theory

(where $\overset{\circ}{-}$ is the *contraction* operator)

P6 implies that the difference between A revised with x and the previously unrevised A is the combination of x and its contradictions $\neg x$. In other words, revision of A involves removal of the contradictions $\neg x$ before addition of x itself, as demonstrated with P1.

(P7) $A \overset{\circ}{+} (x \wedge y) \subseteq Cn(A \overset{\circ}{+} x) \cup \{y\}$ for any theory A

P7 implies that revision of A with more than one set $(x \wedge y)$ is equivalent to A first revised with x and awaiting revision with y. For example, in the revision of {a, b} with $\{\neg a, y\}$, if we first apply $\neg a$ to the original set, we would have $\{\neg a, b\}$, awaiting revision with {y}

(P8) $Cn((A \overset{\circ}{+} x) \cup \{y\}) \subseteq A \overset{\circ}{+} (x \wedge y)$ for any theory A, provided that $\neg y \notin A \overset{\circ}{+} x$

P8 implies that the intermediate result obtained in P7 is equivalent to revising the original A with the combined set $(x \wedge y)$ provided that y does not contradict the intermediate result of A first revised with x.

The above postulates are summarized in the authors' principle of *revision* as a sequence of *contraction* followed by *expansion*, that is, revision being obtained by first eliminating contradictions and then adding incoming propositions:

$(Cn((A \overset{\circ}{-} \neg x) \cup \{x\}))$.

Although Alchourrón et al. (or AGM) have specified postulates against which a revision function can be assessed, they do not define any revision function themselves.

However, coherencist revision functions have used the AGM postulates as a basis to check for correctness.

5.2.4.2 Why use the AGM postulates?

The question arises as to why we want to apply the AGM postulates in our belief revision discussion. First, recall that we agree with the coherencist school of thought that a belief need not be deleted from a revised belief set simply because its justifier has been deleted. In the development of a coherencist belief revision model, it is necessary to have a standard measure against which the scheme can be assessed for correctness. The AGM postulates provide such a standard. As is mentioned in the next section, they are the only standard against which coherencist revision schemes are measured. We may also argue that instead of assessing our scheme using the AGM postulates, we could develop a better standard of evaluation. Although the development of new standards is a good venture, we must state that our goal is not to re-invent the wheel of evaluation criteria. Furthermore, our goal is the development of an effective belief revision scheme rather than the development of a standard of evaluation.

5.2.4.3 The Dalal coherencist revision scheme

Representative of the coherencist approach is the work done by Dalal (1988) who defines a revision function as follows:

$$\psi \circ \mu \quad = \quad \textit{Revise}(\psi, \mu)$$

in which a knowledge base or set of beliefs ψ is revised using incoming beliefs μ. A major principle guiding *Revise* is the *persistence of prior knowledge*, which advocates *minimal change* on the basis of consistency. The principle of persistence of prior knowledge states that

> "As much old knowledge as possible should be retained in the revised knowledge...if $\psi \cup \{\mu\}$ is consistent then $\psi \circ \mu = \psi \cup \mu$"

In other words, the revision of ψ with a noncontradictory $\{\mu\}$ is done by simple addition of μ into ψ (i. e. ψ union μ). However for contradictory $\{\mu\}$ leading to inconsistencies in ψ, the revision process stops deletion of beliefs at the first sign of consistency in ψ, thus maintaining the principle of minimal change. To implement this, Dalal defines functions *res* and $G(\psi)$ as follows:

> Given a formula or belief set ψ and $\{\alpha_1, \ldots \alpha_p\}$ as the set of atoms or beliefs occurring in μ, then
>
> $$\psi \circ \mu = G(\psi) \cup \{\mu\}$$
>
> where $G(\psi) = res_{\alpha_1}(\psi) \vee \ldots \vee res_{\alpha p}(\psi)$
>
> and $res_{\alpha_i}(\psi) = \psi^{+}_{\alpha_i} \vee \psi^{-}_{\alpha_i}$
>
> and $\psi^{+}_{\alpha_i}$ and $\psi^{-}_{\alpha_i}$ are exclusion formulae such that
>
> 1) $\alpha_i \notin \psi^{+}_{\alpha_i}$ and $\alpha_i \notin \psi^{-}_{\alpha_i}$, and
>
> 2) $\psi_{\alpha_i} \approx (\alpha_i \wedge \psi^{+}_{\alpha_i}) \vee (\neg \alpha_i \wedge \psi^{-}_{\alpha_i})$

$res_{\alpha_i}(\psi)$ is called the *resolvent* of ψ with respect to α_i, G is a generalization operator which takes the set ψ and returns a subset of its logical closure, which is the set of resolvents over the beliefs in μ. $\psi^{+}_{\alpha_i}$ and $\psi^{-}_{\alpha_i}$ are local exclusion or contraction formulae which use truth values to remove the contradictions of α_i from ψ. For $\psi^{+}_{\alpha_i}$, α_i is replaced by logical truth ($\mathbb{T}$) in ψ while it is replaced by $\mathbb{F}$ for $\psi^{-}_{\alpha_i}$. The above formulae can be better understood if we look at the following example.

Given $\psi = \{a, \neg b\}$ and $\mu = b$

For convenience, Dalal represents ψ as $(a \wedge \neg b)$

ψ and μ conflict in the truth value of b, so ψ is resolved on b (that is, the belief occurring in μ)

$\psi^{+}_{b} = (a \wedge \neg T) = (a \wedge F) = F$

$\psi^{-}_{b} = (a \wedge \neg F) = (a \wedge T) = a$

$res_b(\psi) = \psi_b^+ \vee \psi_{\neg b}^- = a \vee F = a$

$G(\psi) = res_b(\psi)$

and $\psi \circ \mu = G(\psi) \cup \{\mu\} = a \wedge b$

(Dalal replaces the **UNION** operator $\cup$ by the logical **AND** operator $\wedge$)

The final *Revise* value ($\psi \circ \mu$) obtained removes the contradiction ($\neg b$) from the original knowledge base ψ, and introduces the incoming value b. From the above example, we see that Dalal's $\psi_{\alpha_i}^+$ and $\psi_{\alpha_i}^-$ formulae are a logical implementation of Alchourrón et al.'s principles of *contraction* and *expansion* applied locally to conflicting beliefs. Dalal however uses logical F and T to implement this local contraction and expansion (representing $\psi_{\alpha_i}^-$ and $\psi_{\alpha_i}^+$), respectively, thus excluding the affected contradiction from the resultant resolvent. Globally, however, the collection of resolvents represents the actual contraction of the entire belief base to obtain $G(\psi)$. The final expansion is performed when $G(\psi)$ is combined with the new beliefs μ.

Another coherencist work is that by Boutilier & Goldszmidt (1993) who have defined a revision model $M^*_{A \Rightarrow B} = (W, \leq, \varphi)$ to revise a knowledge base of inference rules or *conditional beliefs* of the form $(A \Rightarrow B)$, where W is a set of worlds (assigned valuations by φ) or possible states of conditionals, and $\leq$ is a *plausibility* ordering over W such that if $v \leq w$ then w is at least as plausible as v. They also apply the principles of contraction and expansion by defining two sub-models $M^-_{A \Rightarrow B}$ which removes negating conditionals of the form $(A \Rightarrow \neg B)$, and $M^+_{A \Rightarrow B}$ which finally adds the incoming conditional $(A \Rightarrow B)$. A similar revision model is defined in Boutilier (1994) but this time based on *epistemic entrenchment* of beliefs (Gardenförs 1984), which means the relative affective willingness of an agent to give up one belief in the presence of another (newer) belief. A high entrenchment value implies that the affected belief is

very important to the agent, and for any two beliefs a and b, $a \leq_E b$ implies that b is at least as epistemically entrenched as b.

There are three major differences between the works done by Boutilier & Goldszmidt, and Dalal. First, Boutilier & Goldszmidt applied plausibility ordering which made them alter the plausibility values of existent conditionals in order to implement expansion and contraction; the plausibility feature makes their approach more applicable to systems like GUIDON, which use certainty factors. Secondly, Boutilier & Goldszmidt worked with inference rules which may be viewed as connected sub-beliefs, while Dalal worked with simple non-implicational beliefs. Thirdly, Boutilier & Goldszmidt applied an optimizing *isolation* technique in which they partitioned the knowledge base KB into KB_I and KB_J where KB_I is called the *minimal complete inconsistent set*, or the set of conditionals which are identified as directly responsible for the negation of an incoming conditional, and KB_J is the set of conditionals which are unrelated or have no contradictory effects on the incoming conditional. The contraction process $M^-_{A \Rightarrow B}$ is therefore performed on KB_I, leaving KB_J untouched, with the resultant KB being $KB = M^-_{A \Rightarrow B}(KB_I) \cup KB_J \cup (A \Rightarrow B)$. Dalal on the other hand applied his resolvent scheme on the entire knowledge base ψ without first separating contradictions. In each case however, the authors showed that their revision models satisfied the AGM postulates.

5.3 Summary

We have introduced the aspects of our student modeling module in this chapter. We have also discussed belief maintenance, representation and revision issues which will affect our design, along with some relevant research done in belief revision. Essentially, the major issues that need to be addressed in our intelligent tutoring model are the following:

- identification of student beliefs (how do we know what the student believes?)

- representation of student beliefs (how can we formally represent beliefs for optimal manipulations and modeling?)
- relationship of beliefs with the Tutor's knowledge base and identification of misconceptions (how can we identify the student's beliefs from our representation of expert knowledge in the knowledge base, and how can we trace misconceptions using the knowledge base?)
- identification of missing beliefs (how do we know which relevant information the student is not applying to the problem at hand, whose absence is keeping the student from obtaining a complete solution?)
- identification of contradictory student beliefs in the face of a new belief
- removal of contradictions and propagation of deletions (how do we choose which beliefs to delete, and what kinds of dependency-directed deletions will be performed to maintain consistency of the student's belief base?)

In the next chapter, we will discuss our approach to the above issues, in the light of the discussions in this chapter. Particularly, we will discuss an improved revision function which also satisfies the Gardenförs postulates. We will also specify algorithms that can be used to obtain the capabilities in our Modeler components.

CHAPTER SIX: MODELER ALGORITHMS AND EXAMPLES

In this chapter we present our belief representation and belief revision approaches in light of the review in Chapter 5. Specifically we address the following issues:

- identification of a belief
- representation of a belief
- representation of a negated belief
- determination of a belief's correctness with respect to the knowledge base
- identification of a belief's contradiction among existent beliefs
- identification of missing beliefs
- maintenance of consistency of the overall belief base (belief revision)
- decision as to which beliefs to delete or keep after removal of contradictions

We also present algorithms which can be used to obtain effective modeling results from components of the Modeler. As mentioned earlier, our prototype implementation is based on the DSM-III-R manual's guidelines laid out for the differential diagnosis of mental disorders. A sample graph based on the manual, showing the differential diagnosis of psychotic disorders, is shown in Appendix C. The Knowledge-base rules we have constructed using that sample are in Appendix D. Our first set of algorithms will illustrate how the expert problem solver applies the knowledge base rules in the solution of a problem. This will enable us have a clearer picture of how a student will be assessed in the same problem-solving procedure. Then we will specify the algorithms for the Knowledge-Finder, Response Analyzer, Deviation-Finder and the Belief-Revision Unit. Each algorithm is explained in detail, and is followed by one or more examples. In all cases, the font conventions used in the algorithms are as follows:

Brush script capital letters ($\mathcal{ABCDEFGH}$ ...) represent:	sets
Symbol characters ($\alpha\ \beta\ \chi\ \delta\ \varepsilon\ \phi\ \gamma$ ...):	set elements
Small bold letters (**procedurename**):	procedure-names

Regular letters: other variable names

6.1 Nature of belief objects

In order to understand our belief revision model, it is important to examine the nature of information to be manipulated with respect to the prototype domain of the diagnosis of psychotic disorders and particularly with respect to the knowledge base defined for the domain. First we are modeling the student's beliefs in terms of cause-effect relationships (that is, given a set of symptoms, what are the student's beliefs about the possible disorders or related symptoms and higher-level conclusions?). A given symptom is the belief's *antecedent* while the student's assertion as to the symptom's effect is the belief's *conclusion*. The student is given an initial set of symptoms (without associated conclusions) and is expected to construct his/her own beliefs based on the initial set. A belief could be observed by the Tutor during interaction or derived using predefined inference rules or variable-relationship predicates. Examples of variable relationships that may be represented in inference rules include implication, equivalence, or negation. For example, suppose we have the following definition (where **imp** is an implication operator):

(light-moods T) **imp** (brief-reactive-psychosis T) (1)

and the student believes that

(catatonic-stupor T) $\Rightarrow$ (light-moods T) (2)

the Tutor can apply (1) and insert a new (derived) belief

(catatonic-stupor T) $\Rightarrow$ (brief-reactive-psychosis T) (3)

In list form, the above three beliefs could be represented as

(**imp**, (light-moods T), (brief-reactive-psychosis T)) (1')
((catatonic-stupor T), (light-moods T)) (2')
((catatonic-stupor T), (brief-reactive-psychosis T)) (3')

It is clear from the above that we represent the student's basic belief as a cause-effect pair. This representation is similar to the implicational molecule discussed by Abelson & Reich (1969) except that each atom represents a variable-value relationship. Also, an inference rule must necessarily contain information as to the kind of inference being made, which is why the **imp** keyword is used to define the relationship between the next two atoms in (1'). Our belief structure is also similar to that used by Boutilier & Goldszmidt (1993), except that we do not apply any plausibility ordering which was central to their own revision scheme.

To represent a negated belief, we assign a **False** value to the item that is negated. This type of representation is facilitated by the fact that diagnosis can be viewed in terms of the presence or absence of a disorder and we have constructed the knowledge base preconditions and conclusions in terms of logical truth values. For example, suppose the student believes that "The presence of catatonic-stupor suggests the absence of light-moods symptoms", a negated version of belief (2') above. This is simply represented by setting the value of light-moods to **False** as follows:

((catatonic-stupor T), (light-moods F)) (4)

Negation of numeric variables is by changing the comparative operator involved, like changing $<$ to $\geq$, for example.

Identification of contradictions therefore becomes a case of evaluating components of two beliefs being compared and determining whether the truth values differ. For example, let us compare beliefs (2') and (4).

((catatonic-stupor T), (light-moods T)) (2')
((catatonic-stupor T), (light-moods F)) (4)

Evaluation of the antecedent of (2') sets the value of catatonic-stupor to **True**. Comparison with the antecedent of (4) would yield a **True** in light of (2'), since

(catatonic-stupor T) = (catatonic-stupor T)

On the other hand, comparison with the conclusion of (4) would yield a **False** in light of the value set by the evaluation of the conclusion in (2'). This indicates that belief (4) is a contradiction of belief (2'). This example compares beliefs whose variable names are similar. For beliefs with different variable names, we would need to search for inference rules which define the relationship between the two variables being compared. If an inference rule exists which identifies an equivalence relationship between the two variables, for instance, then they would be compared as though they had the same name.

Determination of a belief's correctness with respect to the knowledge base is done by looking for the belief's antecedent in the preconditions of knowledge base rules, and chaining rules to determine if the belief's conclusion can be found in the conclusion of any of the chained rules. If so, the belief would be labeled as correct, and incorrect otherwise. This labeling is done by the Deviation Finder.

6.2 Belief structure

The belief structure in this model is a quintuple with the following components:

(*id, type, antecedent, conclusion, justifier-set*)

Meanings of components are as follows:

id:	belief's unique identifier
type:	type of belief
antecedent:	belief's antecedent or preconditions (cause)
conclusion:	belief's conclusion (effect of antecedent)
justifier-set:	set of beliefs from which belief is derived

Id is a unique identifier for each belief. We assume integer identifiers (≥ 1).

The *type* of a belief could be OBSERVED (that is, explicitly specified by the student during interaction), *DERIVED* (that is, derived by Tutor using inference rules), or *ASSUMED* (that is, assumed by the Tutor as being a likely student belief, with no underlying justification). Observed beliefs are obtained during dialogs between the

Tutor and the student. These dialogs can be done using a structured communication system which avoids ambiguity.

Antecedent and *conclusion* in the belief structure are the basic components of a belief, referring to a symptom and its associated effect as perceived by the student. Note that the conclusion in one belief can serve as the antecedent in another belief, seeing we have a hierarchically organized knowledge-base where there can be several intermediate conclusions before arrival at a final disorder. *Justifier-set* is the set of beliefs and inference rules from which a belief is derived. The justifier-set for inference rules and observed beliefs is null.

6.3 Revision Preview: An example

The essential processing of beliefs that will be done by the Student Modeler are the identification of beliefs (by the Response Analyzer), labeling of beliefs and identification of misconceptions (by the Deviation Finder), and maintenance of the belief base consistency (by the Belief Revision Unit). In this section, we give an example of how a culprit belief is identified in the event of a contradiction, how to trace through a belief base to identify beliefs dependent on the culprit, and how the model determines what to do with those dependent or derived beliefs.

Let us assume the following belief set (this time, we shall include details of each belief, for completeness):

(INF1, **imp**, (strong-moods T), (schizophrenic-symptoms T)) (5)
(1, OBSERVED, (visual-hallucinations T), (strong-moods T), { }) (6)
(2, DERIVED, (visual-hallucinations T), (schizophrenic-symptoms T), {1, INF1}) (7)
(3, OBSERVED, (visual-hallucinations T), (strong-moods F), { }) (8)

"imp" in (5) signifies an implication. First, we separate inference rules into an INFERENCE-RULEBASE, since they are not student beliefs in themselves, but rules used to derive new beliefs from older student beliefs. That leaves beliefs (6) to (8) in

the belief base. INF1 in the inference rule is the unique identifier indicating that as the first inference rule. Belief ids are 1, 2 and 3 for the student beliefs, so we shall renumber them for ease of reference in this discussion.

(1, OBSERVED, (visual-hallucinations T), (strong-moods T), { }) (1)
(2, DERIVED, (visual-hallucinations T), (schizophrenic-symptoms T), {1, INF1}) (2)
(3, OBSERVED, (visual-hallucinations T), (strong-moods F), { }) (3)

Justifier sets for observed beliefs 1 and 3 are empty (since they are explicitly displayed by the student) while the justifier set for the derived belief 2 includes the observed belief 1 and the inference rule INF1. Let us assume that belief 3 has just been newly observed. We notice that this new belief is contradicted by the older observed belief 1 (and subsequently the derived belief 2). So for the incoming belief 3, we identify belief 1 as a culprit contradiction (using evaluation of truth values described in Section 6.1), and we should delete belief 1. Since belief 2 is derived from 1, it is logical to delete it as well, but suppose that the Tutor queries the student and the student actually believes that (visual-hallucinations T) $\Rightarrow$ (schizophrenic-symptoms T) even though the justifying belief has been deleted. In our scheme, we constrain the belief-revision unit to retain belief (2) and to change its status to an observed status as follows:

(2, OBSERVED, (visual-hallucinations T), (schizophrenic-symptoms T), { })

We are therefore left with the following:

Belief base:
(2, OBSERVED, (visual-hallucinations T), (schizophrenic-symptoms T), { }) (2)
(3, OBSERVED, (visual-hallucinations T), (strong-moods F), { }) (3)

Inference-rulebase:

(INF1, **imp**, (strong-moods T), (schizophrenic-symptoms T)) (INF1)

6.4 Formalized revision scheme

In this section we present a formal discussion of the revision scheme we have employed in the preceding example. First we are using the coherencist belief revision approach, that is, a derived belief is not deleted simply because its justifier has been deleted. Instead, the student is queried to determine whether those derived beliefs have actually been disbelieved or not. Results of these queries would determine what would be done with the affected derived beliefs. As mentioned in Chapter 5, the coherencist approach is more practical than the foundationalist approach in a tutoring application. Secondly, because we are not applying certainty factors, we will employ the resolvent revision scheme developed by Dalal (1988). Surprisingly, little or no work has been built on the Dalal scheme, due largely to the fact that most belief revision research has been focused on the TMS/ATMS foundationalist models. We will present modifications to the Dalal scheme which will better optimize the revision process. We shall compare results obtained using our improved scheme with results from the Dalal scheme and the Huang et al. scheme which is representative of the foundationalist revision approach. We shall also show that this improved scheme satisfies the AGM postulates which have been the basis for evaluating coherencist revision schemes.

6.4.1 Failure of the Dalal-resolvent scheme

We introduce our revised scheme with an example using the Dalal resolvent scheme.

Let us assume an initial belief set taken from Huang et al. (1991):

$\{A, B, S, \neg T, U, V\}$

Since we are using T for logical TRUTH, we shall replace T with L, for readability. We therefore have an initial belief set $\{A, B, S, \neg L, U, V\}$. This belief set is to be updated by incoming beliefs $\{C, L, \neg U\}$

Using the Dalal scheme,

$\psi = \{A, B, S, \neg L, U, V\}$

$$= \quad \{A \wedge B \wedge S \wedge \neg L \wedge U \wedge V)$$

$$\mu = \quad \{C, L, \neg U\}$$

$$= \quad \{C \wedge L \wedge \neg U\}$$

ψ and μ conflict in the truth-values of L and U, so we need to resolve ψ with L and U only.

$$res_L(\psi) = \psi_L^+ \vee \psi_L^-$$

$$= \{A \wedge B \wedge S \wedge \neg \mathbb{T} \wedge U \wedge V\}$$
$$\vee$$
$$\{A \wedge B \wedge S \wedge \neg \mathbb{F} \wedge U \wedge V\}$$

$$= \{\mathbb{F}\} \vee \{A \wedge B \wedge S \wedge U \wedge V\}$$

$$= \{A \wedge B \wedge S \wedge U \wedge V\}$$

Notice that the resolvent operation is done using the variable name involved in the conflict, and not the value of the incoming belief value itself (which is added after resolvents have been obtained), as will be better illustrated by the next resolvent.

$$res_U(\psi) = \psi_U^+ \vee \psi_U^-$$

$$= \{A \wedge B \wedge S \wedge \neg L \wedge \mathbb{T} \wedge V\}$$
$$\vee$$
$$\{A \wedge B \wedge S \wedge \neg L \wedge \mathbb{F} \wedge V\}$$

$$= \{A \wedge B \wedge S \wedge \neg L \wedge V\} \vee \{\mathbb{F}\}$$

$$= \{A \wedge B \wedge S \wedge \neg L \wedge V\}$$

$$G(\psi) = res_L(\psi) \vee res_U(\psi)$$

$$= \{A \wedge B \wedge S \wedge U \wedge V\} \vee \{A \wedge B \wedge S \wedge \neg L \wedge V\}$$

$$= \{A \wedge B \wedge S \wedge U \wedge \neg L \wedge V\}$$

and $\psi \circ \mu = G(\psi) \wedge \{\mu\}$

$$= \{A \wedge B \wedge S \wedge U \wedge \neg L \wedge V\} \wedge \{C \wedge L \wedge \neg U\}$$

$$= \{\mathbb{F}\},$$

since (L and $\neg$L} and (U and $\neg$U} both yield $\mathbb{F}$.

The above example illustrates that Dalal's revision scheme fails in a case of multiple contradictions of the form {x, y} where $\{\neg x, \neg y\} \in \psi$, as shown. This failure arises

from the generalization operator $G(\psi)$ which OR's individual resolvents before the incoming belief set μ is introduced. The OR operation on resolvents re-introduces all existent contradictions which lead to logical **False** when ANDed with μ.

Secondly, we notice that Dalal does not apply the resolvent function on non-contradictory beliefs like C above, but he defined $G(\psi)$ as a generalization operator applied to resolvents of all elements in μ, which is a misleading definition.

Thirdly, Dalal allows the set-theoretic union operator ($\cup$) to be logically equivalent to an AND operator, "when convenient". Essentially however, the resolvent process is an intertwined procedure of first viewing beliefs as a set of logical variables to allow the AND and OR operations and replacement of conflicting variables by **T** or **F**, then *releasing* the variables from the logical mode and restoring them to set elements to allow the union operation with the appropriate incoming belief. We believe that it is not necessary to mingle logical and set-theoretic operations in this scheme essentially because one of Dalal's ψ_α^+ and ψ_α^- always yields a **False** while the other always yields $\{\psi - \{\neg\alpha\}\}$. Since $\psi_\alpha^+ \vee \psi_\alpha^- = \{\psi - \{\neg\alpha\}\} \quad \forall\ \alpha$, it is redundant to perform the logical operation $\psi_\alpha^+ \vee \psi_\alpha^-$. This would eliminate the need for a logical operation since the other functions in the Dalal scheme are set-theoretic.

6.4.2 An improved belief revision scheme

In view of the failure of the Dalal resolvent scheme as illustrated by the last example, we have developed a revision scheme with the following modifications to the Dalal scheme:

- separates incoming contradictions and incoming non-contradictions as was done by Boutilier and Goldszmidt (1993)
- replaces the logical equation $res_\alpha(\psi) = \psi_\alpha^+ \vee \psi_\alpha^-$ by the equivalent set-theoretic equation $res_\alpha(\psi) = \{\psi - \{\neg\alpha\}\}$
- performs recursive resolution to avoid re-introduction of contradictions

Therefore, given a belief base ψ and a set of incoming beliefs $\mu = \{\mu_1, \ldots \mu_n\}$, we define an isolation scheme that separates incoming contradictions and non-contradictions as follows:

$C(\mu) = (\{\alpha_1 \cup \ldots \cup \alpha_p\} \mid \neg\alpha_i \in \psi, \forall i)$

(that is, the set of incoming beliefs in μ which contradict ψ).

and $NC(\mu) = (\{\eta_1 \cup \ldots \cup \eta_m\} \mid \neg\eta_k \notin \psi, \forall k)$

(that is, the set of incoming beliefs in μ which do not contradict ψ).

Next we define a *recursive* revision function $\Re_{\mu}^{\psi}$ which revises ψ with μ, such that

$$\Re_{\mu}^{\psi} = \mathscr{C}_{\mu}^{\neg\psi} \cup NC(\mu) \qquad \text{(I)}$$

where

$\mathscr{C}_{\mu}^{\neg\psi}$ is a contradiction removal function such that

$$\mathscr{C}_{\mu}^{\neg\psi} = res_{\alpha_p}(\psi) \qquad \text{(II)}$$

and *res* is a recursive version of Dalal's resolvent such that

$$res_{\alpha_1}(\psi) = \{\psi - \{\neg\alpha_1\}\} \cup \{\alpha_1\} \qquad \text{(III)}$$

and

$$res_{\alpha_i}(\psi) = \{(res_{\alpha_{i-1}}) - \{\neg\alpha_i\}\} \cup \{\alpha_i\}, \ \forall i \geq 2 \qquad \text{(IV)}$$

Our resolvent function is a *resolvent of resolvents* in that it resolves multiple contradictions recursively by iteratively removing an existent contradictory belief, adding the appropriate replacement in (III), and performing the same operation using the next incoming contradicted belief on the result obtained in the preceding iteration (IV). $\mathscr{C}_{\mu}^{\neg\psi}$

thus performs contraction of the existent knowledge base by removing all identified contradictions using $C(\mu)$, as well as partial expansion by the addition of affected new beliefs. The remaining expansion is done by $\Re_{\mu}^{\psi}$ itself when non-contradictory beliefs are added to the belief base. The recursive removal of new contradictions from the preceding resolvent (and not from the original belief base) ensures that all contradictions are actually removed and that no conflict arises when an incoming belief is added, which problem occurred with the Dalal approach.

6.4.3 Example

We will now illustrate the above scheme using the last example. Given ψ and μ as follows:

ψ = {A, B, S, ¬L, U, V}
μ = {C, L, ¬U}

ψ and μ conflict in the truth-values of L and ¬U, so we need to resolve ψ with L and ¬U.

$C(\mu)$ = {L, ¬U}

$NC(\mu)$ = {C}

$res_L(\psi)$ = $\{\psi - \neg L\} \cup \{L\}$

= {A, B, S, U, V, L}

This result is the one used for the resolvent with ¬U, not the original belief base ψ. Notice that our resolvent function is being performed using the actual value of an incoming belief, not just the variable name(s) involved.

$res_{\neg U}(\psi)$ = $\{(res_L(\psi)) - \{\neg(\neg U)\}\} \cup \{\neg U\}$

= $\{\{A, B, S, U, V, L\} - \{U\}\} \cup \{\neg U\}$

= {A, B, S, V, L, ¬U}

$$\therefore \mathscr{e}^{-\Psi}_{\mu} = \text{res}_{\neg U}(\psi) = \{A, B, S, V, L, \neg U\}$$

Now that resolvents have been computed for all contradictions, we add the non-contradictory incoming beliefs, NC(μ)

$$\begin{aligned} \Re^{\Psi}_{\mu} &= \mathscr{e}^{-\Psi}_{\mu} \cup \text{NC}(\mu) \\ &= \{A, B, S, V, L, \neg U\} \cup \{C\} \\ &= \{A, B, S, V, L, \neg U, C\}. \end{aligned}$$

This revised scheme does not fail in the face multiple contradictions. Also, it is a less restrictive scheme which produces a belief set that is larger than Huang et al.'s foundationalist approach that yielded {B, C, L, ¬U} and deleted {A, S, V}. This brings us to the issue of belief dependency. Huang et al. identified S and V as beliefs derived from the deleted contradictions while A was an observed belief. Yet A was deleted in their scheme along with the derived S and V. Dalal on the other hand did not define any dependencies among his beliefs.

6.4.4 Extensions on dependencies

From discussions in Chapter 5, we recognize that beliefs do have dependencies, although we agree with coherencists that derived beliefs do not necessarily have to be deleted simply because their justifiers have been deleted. For our tutoring model therefore, we extend Dalal's definition of persistent prior knowledge as follows:

Definition: Persistence of dependent beliefs

> As much old knowledge as possible should be retained in the revised knowledge...if $\psi \cup \{\mu\}$ is consistent then $\psi \circ \mu \approx \psi \cup \mu$. Also let δ be the set of beliefs derived from deleted contradictions (which are underlying prerequisites) and β be the set of other, unrelated beliefs. (Thus $\psi = \delta \cup \beta$), and let the resolution of ψ be performed on β and appropriate elements in δ be reintroduced into ψ after a query process.

Therefore in the presence of incoming beliefs μ, the revision process will be as follows:

$$\Re_{\mu}^{\Psi} = \Re_{\mu}^{\beta} \cup \delta$$
$$= \mathscr{C}_{\mu}^{\neg\beta} \cup \mathrm{NC}(\beta) \cup \delta$$

For the derived belief set δ, we initialize δ_{sc} which is the set of derived beliefs whose status is changed from "DERIVED" to "OBSERVED". We then update δ_{sc} as follows:

$\forall \alpha \in \delta, \quad \delta_{sc} := \delta_{sc} \cup \alpha$

iff $\exists \neg\alpha$ observed in query process.

This means that if the student does not exhibit a conflicting belief, an affected derived belief is retained in the belief base with its status changed. The revised belief base therefore becomes

$$\Re_{\mu}^{\Psi} = \Re_{\mu}^{\beta} \cup \delta_{sc}$$
$$= \mathscr{C}_{\mu}^{\neg\beta} \cup \mathrm{NC}(\beta) \cup \delta_{sc}$$

The best case result would be obtained when the student still holds all derived beliefs, resulting in {A, B, S, V, L, ¬U, C} from our last example above, while the worst case is where no derived belief is re-introduced, (or $\delta_{sc} = \varnothing$), resulting in {A, B, L, ¬U, C}. In either case, while maintaining belief system consistency, this revision scheme is less restrictive than Huang et al.'s foundationalist approach in which the non-contradicted observed belief A was deleted.

6.4.5 Extensions on dependents of incoming beliefs

Further extension is required before the definition of $\Re_{\mu}^{\Psi}$ can be complete. It is surprising to note that in Huang et al.'s revision example, the incoming beliefs are not accompanied by dependents or derived beliefs. In order for a revision process involving belief dependencies to be complete, revision with incoming beliefs μ must be followed

by revision with beliefs which are derivable from the beliefs in μ. Revision therefore becomes a recursive process of revising with new beliefs, their dependents, their dependents' dependents, until no new dependents are found. At best, this recursive process would result in the simple union of a previously revised belief set and a derived belief, λ. At the worst, the process would involve the removal of the dependent's negation $(\neg\lambda)$ before the dependent itself is added. We therefore further extend the revision function as follows:

$$\Re_{\mu}^{\Psi} = \Re_{\lambda}^{\Re_{\mu}^{\Psi}} \quad \forall\, \lambda \text{ derivable from } \mu.$$

Therefore if $\Re_{\mu}^{\Psi} = \mathscr{C}_{\mu}^{\neg\beta} \cup NC(\beta) \cup \delta_{sc}$

$$= \varphi,$$

$$\Re_{\lambda}^{\Re_{\mu}^{\Psi}} = \Re_{\mu}^{\varphi}$$

$$= \mathscr{C}_{\mu}^{\neg\varphi} \cup NC(\varphi) \cup \delta'_{sc}$$

where δ'_{sc} is the set of beliefs derived from the contradictions of λ. Again, this is a recursive process in that if $\Re_{\lambda}^{\Re_{\mu}^{\Psi}}$ is γ, say, we would need to revise γ based on its own set of derived beliefs.

A final note: Although our beliefs are of the form $A \Rightarrow B$, it is not difficult to see how a conflicting belief is identified. From basic logic, $A \Rightarrow B$ is **False iff** B is **False** when

A	B	$A \Rightarrow B$
T	T	T
T	F	F
F	T	T
F	F	T

A is True, as shown in the truth table above Therefore if a basic belief is X = ((catatonic-stupor T) (light-moods T)), the task of finding ¬X would be that of determining whether an incoming belief evaluates to ((catatonic-stupor T) (light-moods F)). If this is the case, then we would have succeeded in concluding ¬X.

6.4.6 $\Re_{\mu}^{\Psi}$ satisfies the AGM postulates

It now remains to show that $\Re_{\mu}^{\Psi}$ satisfies the AGM postulates, as should be the case for any coherencist revision scheme. We shall illustrate postulate satisfaction with examples. For all cases, let us assume that A = {D, B, S, ¬L, U, V}, and for demonstration purposes, let us assume that no beliefs are derived from incoming beliefs. Please recall that these postulates and their meanings have been discussed in Section 5.2.4.

Postulate P1: $A \overset{\circ}{+} x$ **is always a theory**

As defined earlier, a theory is a set of beliefs closed under its consequence. In our belief set, derived beliefs are stored along with their justifying beliefs, so the consequence of the belief set is the belief set itself.

If A = {D, B, S, ¬L, U, V} and x = C,

$A \overset{\circ}{+} x = \Re_{C}^{A}$ = {D, B, S, ¬L, U, V, C}

which is also a theory. This is one case when the incoming belief x is non-contradictory, or $\neg x \notin A$. When x is contradictory, revision is by first deleting the contradiction before adding x, as shown in the example for P3. In either case, the resultant belief set is consistent.

Postulate P2: $x \in A \overset{\circ}{+} x$

Naturally, a belief set revised with a new belief must contain the belief with which it is revised. Thus $C \in \Re_C^A$ as shown in the last example. The most fundamental inclusion test for consistency is that of addition into an empty belief set, or $\varnothing \overset{\circ}{+} x$. We show that revision of a null set with a non-null belief yields a consistent belief set. Since $\neg x \notin \varnothing$ $\forall x$, $\varnothing \overset{\circ}{+} x = \{x\}$, $\forall x$, which is a consistent result.

Postulate P3: **If $\neg x \notin Cn(A)$, then $A \overset{\circ}{+} x = Cn(A \cup \{x\})$**

If x does not contradict A, then revision of A with x is the union of x with A. For example, if $A = \{D, B, S, \neg L, U, V\}$ and $x = C, \neg C \notin A$,

$\therefore \Re_C^A = A \cup C = \{D, B, S, \neg L, U, V, C\}$. This is true regardless of the value of x.

Postulate P4: **If $\neg x \notin Cn(\varnothing)$, then $A \overset{\circ}{+} x$ is consistent under Cn**

Revision with a non-empty set of beliefs produces a consistent belief base. For example, $\neg C \notin \varnothing$, therefore $\Re_C^A = A \cup C = \{D, B, S, \neg L, U, V, C\}$ which is a consistent (unconflicted) belief set. The base test for this postulate is when all elements in A are recursively deleted as a result of the introduction of a new belief, x. In this case, the resulting set is {x}, which satisfies postulate P2 and is consistent as is required by P4.

Postulate P5: **If $Cn(x) = Cn(y)$, then $A \overset{\circ}{+} x = A \overset{\circ}{+} y$**

This is a case of coreferentiality. Revision with identical values yield identical results. For instance, if we have two beliefs named x and y, and x = y = (all birds can fly), then $\Re_x^A = \Re_y^A = \{D, B, S, \neg L, U, V,$ "all birds can fly"$\}$

Postulate P6: $(A \overset{\circ}{+} x) \cap A = A \overset{\circ}{-} \neg x$, **whenever** A **is a theory**

As previously defined, $\overset{\circ}{-}$ is the contraction operator signifying the result of the removal of a negating belief from a belief base. In our scheme this is $\mathscr{C}_{\mu}^{-\psi}$ without the partial expansion by μ. For example, if A = {D, B, S, ¬L, U, V} and x is L,

$A \overset{\circ}{-} \neg x = \{D, B, S, U, V\}$

$A \overset{\circ}{+} x = \Re_{x}^{A} = \{D, B, S, U, V, L\}$

and $(A \overset{\circ}{+} x) \cap A = \{D, B, S, U, V, L\} \cap \{D, B, S, \neg L, U, V\}$

$= \{D, B, S, U, V\}$

$= A \overset{\circ}{-} \neg x$

Postulate P7: $A \overset{\circ}{+} (x \wedge y) \subseteq Cn(A \overset{\circ}{+} x) \cup \{y\})$ **for any theory** A

Let A = {D, B, S, ¬L, U, V}

x = L, and

y = ¬ U

$A \overset{\circ}{+} (x \wedge y) = \Re_{\{L \wedge \neg U\}}^{A} = \{D, B, S, V, L, \neg U\}$

$(A \overset{\circ}{+} x) = \Re_{L}^{A} = \{D, B, S, U, V, L\}$, and

$(A \overset{\circ}{+} x) \cup \{y\} = \{D, B, S, U, V, L, \neg U\}$

$\therefore A \overset{\circ}{+} (x \wedge y) \subseteq (A \overset{\circ}{+} x) \cup \{y\}$

This result can be easily explained since $(A \overset{\circ}{+} x) \cup \{y\}$ has not been revised with y and thus contains {y} as well as all contradictions of y while $A \overset{\circ}{+} (x \wedge y)$ has been revised with y and contains {y} but not any contradictions of y.

Postulate P8: $Cn((A \dot{+} x) \cup \{y\}) \subseteq A \dot{+} (x \wedge y)$ **for any theory A,**

provided that $\neg y \notin A \dot{+} x$

This postulate is related to P7. Using the last example, suppose $y = C$ and $\neg y \notin A \dot{+} x$, it follows that $((A \dot{+} x) \cup \{y\}) \subseteq A \dot{+} (x \wedge y)$

since $\{D, B, S, U, V, L\} \cup \{C\} \subseteq \{D, B, S, U, V, L, C\}$

Of course the reason for assuming a subset relationship is to allow for beliefs that might be derived from y. If other beliefs are derivable from y, then $(A \dot{+} (x \wedge y))$ would be larger than $((A \dot{+} x) \cup \{y\})$.

6.4.7 Single versus double belief bases

Finally we consider the issue of whether beliefs should be stored in a single belief base or in two belief bases separated on the basis of correctness labels. Since we are working on a tutoring application, labeling beliefs as correct or incorrect is a necessary task, for modeling purposes. The issue then is whether or not correctness labels should be applied in belief revision. It is natural to expect that in the belief revision process, an incoming correct belief would be negated by existent incorrect beliefs, and vice-versa. If beliefs are stored without any correctness labels, identifying contradictions would involve looking at every belief in the entire belief base to determine the beliefs that contradict an incoming belief. On the other hand, if beliefs are labeled, a revision scheme need only look at a negating group for each type of incoming belief. For example if an incoming belief is labeled as correct, the revision scheme need only examine incorrect beliefs in search of contradictions. By this we argue that correctness labels would facilitate the belief revision process in general. Further, since identification of belief labels would involve some computer processing time (we conveniently divide

the lookup of a belief into two phases: identification of correctness label and identification of the belief proper), we also recommend the separation of beliefs into two belief bases based on their correctness labels. This would save the time required to examine each belief in order to determine its label. Therefore, if for instance there are a total of 2n student beliefs, separated into two labeled belief bases, identifying a contradiction would save $2n\theta$ times where θ is the time required to identify a belief label in a single-belief-base environment, which time would not be spent if the beliefs are separated in the first place.

6.4.8 Summary of belief revision

In the foregoing sections we have presented the nature of our belief system along with a new belief revision scheme based on the Dalal-resolvent scheme. The function we have defined is $\Re_{\mu}^{\Psi}$ which is set-theoretic, does not fail in a case of multiple incoming beliefs, and is less restrictive than the Huang et al. scheme which is representative of the foundationalist revision model. We have also shown that $\Re_{\mu}^{\Psi}$ satisfies the AGM postulates and thus qualifies as a coherencist revision scheme. The algorithms that can be used to obtain the effects discussed above are described in the rest of this chapter.

6.5 Overall Modeler activity and underlying assumptions

Since the Modeler is in a multi-module model and our research is not focused on the other modules, we must make some assumptions about aspects of these modules which may affect the Modeler's functions. Figure 6.1 shows the overall flow of activity within the Modeler and how the Modeler connects to the Tutor at the end of the modeling process.

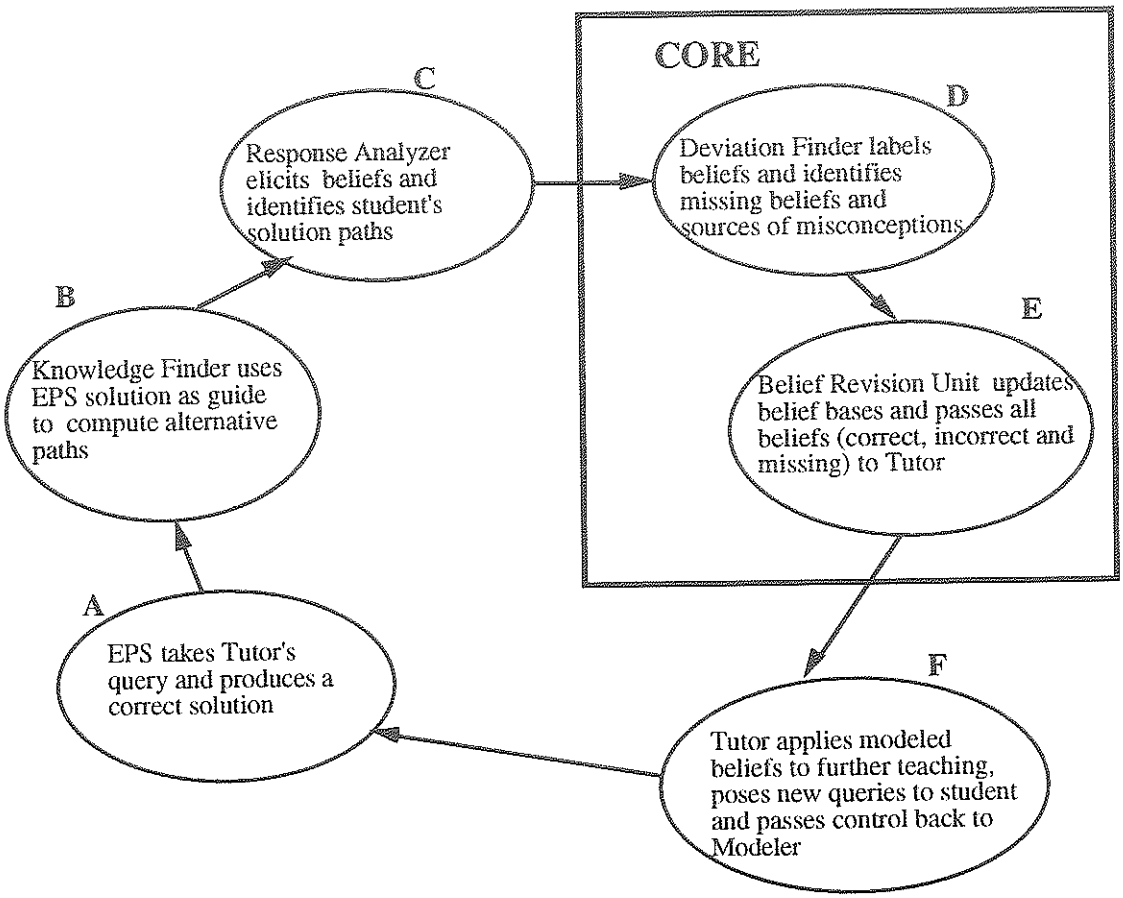


Figure 6.1 Overall flow of Modeler functions: the modeling cycle

First, before student modeling can occur, the student must be presented with a problem. We assume a problem-solving scenario in which a student using the overall tutoring model is presented with a problem and is asked to provide the Tutor with a differential diagnosis based on symptoms provided by the Tutor with the assistance of the Dynamic Questions Module. After a case is presented to the student, the following events follow in sequence:

- the Expert Problem Solver (EPS) performs a forward chain through the knowledge base and solves the same problem. The solution obtained by the Expert Problem Solver will serve as a guide for the Knowledge Finder to generate alternative solution paths.

- the Knowledge Finder performs a backward chain through the knowledge base, using the EPS's diagnosis to generate alternative paths to the problem provided by the Tutor. These alternative paths enable the Modeler to perform strategy-independent student modeling, since the student is not restricted to the resident expert's solution path. The paths obtained at this stage will be analyzed by the Response Analyzer to determine which paths the student is following in his/her solution.

- the Response Analyzer then takes the student through a query session with the aim of constructing beliefs using the student's responses to queries. Using these beliefs, the Response Analyzer examines the alternative paths computed by the Knowledge Finder and selects candidate paths that the student might be applying, based on belief frequencies. For the query process, we assume an appropriate communication system in the user interface which would control the interaction between the Response Analyzer and the student. We also assume that the query process elicits all of the student's beliefs for the current problem. The beliefs obtained during this phase will be labeled by the Deviation Finder and also processed for identification of misconceptions.

- the Deviation Finder, whose central objective is to identify misconceptions, takes all beliefs and traces the knowledge base to determine whether each belief is correct or incorrect. For each incorrect belief, the Deviation Finder then computes chains using the belief's parts and analyzes the chains in order to identify possible misconceptions. The Deviation Finder also identifies missing beliefs by determining which correct beliefs in the EPS solution are missing from the student's

beliefs. Labeled beliefs are passed on to the Belief Revision Unit for belief maintenance.

- the Belief Revision Unit removes beliefs contradicting incoming beliefs, maintains correct-usage counts for correct beliefs, and also derives new more beliefs using inference rules. We have repeatedly mentioned that the belief system will be able to derive new beliefs from observed beliefs using inference rules. We assume that these inference rules exist, and that the belief revision unit simply applies the rules as required.

- the updated belief system is then passed on to the Tutor, which structures further interaction with the student based on the input from the Modeler. Further interaction may involve the Tutor teaching the student to address the missing beliefs, or asking further questions, referring to remedial learning material, referring to textbooks, and so on. At the end of the teaching interaction, the Tutor may present the student with another problem, which reactivates the modeling cycle and the process is repeated.

Details of the above sequence will be presented in the rest of the chapter. Units' algorithms are presented in the same sequence of the overall activity described above: Expert Problem Solver, Knowledge Finder, Response Analyzer, Deviation Finder, Belief Revision Unit. As indicated in Figure 6.1, the core of our work concentrates on the Deviation Finder and the Belief Revision Unit, using beliefs generated by the Response Analyzer.

6.6 Expert Problem Solver's reasoning process

The rules in Appendix D are applied by the Expert-Problem-Solver in the solution of a problem, that is, the generation of a differential diagnosis given a set of symptoms. Two

alternative approaches were considered. The first involved using the tree structure to generate a diagnosis. This however imposes an ordering on the Tutor's presentation of symptoms. For instance, to accurately diagnose DELUSIONAL-DISORDER, the Tutor would have to provide duration data before manic-syndrome. If this order is not followed, there would be a generation of several temporary solution sets until the necessary data are provided. For instance, if manic-syndrome information is provided before duration, then the EPS would have two sets of differential diagnosis:

{DELUSIONAL-DISORDER,
PSYCHOTIC-MOOD-DISORDER,
PSYCHOTIC-DISORDER-NOS}
and {PSYCHOTIC-DISORDER-NOS,
SCHIZOPHRENIA,
SCHIZOPHRENIFORM-DISORDER,
SCHIZOAFFECTIVE-DISORDER,
PSYCHOTIC-MOOD-DISORDER}

As more data is provided, the EPS would then discard unlikely sets. This approach proved to be inefficient, particularly in a situation where the Tutor ultimately does not provide the required symptoms or data. The EPS in such a case would produce multiple differential diagnoses, which would make pedagogy more undirected and confusing to the student.

The second approach is to allow the EPS to use the knowledge-base rules, and perform set unions or intersections as data are provided. This approach does not impose any ordering on data presentation, and is perfectly accurate even in the face of incomplete data. The EPS would use the rules which apply to a current data, and merge the resultant differential-diagnosis group with the previously obtained set. There is only one differential diagnosis at any point in time, and we observed each diagnosis to be accurate with respect to the tree structure, at any point in the solution process. We chose this approach above the tree-reasoning approach.

We have employed the following disorder abbreviations: for this and other algorithms:

DELUSIONAL-DISORDER	DED
PSYCHOTIC-DISORDER-NOS	PDN
PSYCHOTIC-MOOD-DISORDER	PMD
SCHIZOPHRENIA	SCH
SCHIZOAFFECTIVE-DISORDER	SFD
SCHIZOPHRENIFORM-DISORDER	SPD
BRIEF-REACTIVE-PSYCHOSIS	BRP
ORGANIC-MENTAL-DISORDER	OMD

We have implemented the EPS in this design, and the following four algorithms (6.1 to 6.4) describe how this was done.

Algorithm 6.1: The EPS Differential diagnosis

Input: $\mathcal{S}$: Set of symptoms
$\mathcal{KB}$: set of rules in knowledge-base

Process: Forward-chain through $\mathcal{KB}$ to obtain PD for each symptom
Intersect PDs to obtain diagnosis

Variable names:
DD: Differential diagnosis
PD: Partial diagnosis (for a symptom)
resp$: a string response to a query
i, j, count: loop variables
WM: working memory
nsymptoms: number of symptoms
new-prec: current precondition under search
searchbase: subset of $\mathcal{KB}$ being traced
branch-point: a point of branch among rules in searchbase
CDD: current differential diagnosis after considering a new symptom
Mark: a binary flag which indicates whether a rule has been visited by the depth-first algorithm
σ: a symptom in $\mathcal{S}$
τ: a rule in $\mathcal{KB}$

Output: DD_{EPS} (Expert Problem Solver's differential diagnosis)

```
procedure compute-eps-diag( )
  begin
    DD_EPS = WM := ∅                    {initially nulls}              (1)
    count := 0                          {while loop counter}
    while resp$ <> 'no' do                                             (2)
      begin
        input ('any new symptoms? ), resp$                             (3)
        if resp$ <> 'yes' exit ; {while}                               (4)
        nsymptoms := 0                  {initialize number of symptoms}
```

```
count := count + 1                          {increment number of while loops}
for each i := 1 till done do                {Collect Symptoms}                    (5)
  begin
    read σ_i
    append to 𝒮 in WM                       {working memory}
    nsymptoms := nsymptoms + 1
  end {for i}
for each σ_i ∈ 𝒮 do                         {Compute Symptoms' pds}               (6)
  begin
    initialize PD_σi := ∅                                                         (7)
    new-prec := σ_i
    branch-point := 0
    searchbase := 𝒦ℬ
    for each rule τ_j in searchbase do                                            (8)
        Mark_τj := 0
    *chain-rules*(new-prec, searchbase)                                           (9)
  end; {for σ_i}
loop-differential(nsymptoms)                                                      (10)
if count = 1                                                                      (11)
    then  DD_EPS := CDD_nsymptoms
    else  DD_EPS := DD_EPS ∩ CDD_nsymptoms
                                            {Intersection across symptoms}
print DD_EPS                                {Print and reset WM}                  (12)
WM := ∅                                                                           (13)
end; {while}
end; {compute-eps-diag}
```

Algorithm 6.1 obtains the EPS-differential diagnosis given a set of symptoms and the Knowledge-Base rules. We are using the set theoretic covering model approach as used in Reggia et al. (1983). Given a set of symptoms ($\{\sigma_1, \sigma_2, \ldots \sigma_{nsymptoms}\}$) and disorders ($\{\delta_1, \delta_2, \ldots \delta_{ndisorders}\}$), the covering model obtains a minimal set of disorders that commonly account for the given symptoms. This minimal *explanation* is the differential diagnosis (DD_{EPS}), which could be a single disorder or multiple disorders, and is obtained by performing a set intersection across the partial diagnoses (PD) obtained for each separate symptom. In the event that the differential diagnosis set is reduced to a single disorder, it may be called simply a "final diagnosis", and not

necessarily a differential diagnosis, since the latter term is more commonly used for multiple disorders in the same set.

Algorithm 6.1 uses Algorithms 6.2 to 6.4. At the start of the algorithm, DD_{EPS}, WM and each PD, are nulls (1). The algorithm first prompts for the presence of new symptoms (2) - (4), without which the procedure cannot continue. The symptoms are then read into WM (5). Then for each symptom, there is a forward-chain through the Knowledge-Base (KB) to obtain the PD for that symptom(6)-(10). The chaining is done by a recursive procedure named ***chain-rules*** (Algorithm 6.2) which identifies the disorders covered by the current symptom (9). To do this effectively, the variable Mark is initialized to zero for each rule, indicating that no rule has been visited for the current symptom. The PDs resulting from ***chain-rules*** are then sent to a recursive procedure named **loop-differential** (Algorithm 6.4.) which obtains the differential diagnosis at the presence of each new symptom (10). The result of **loop-differential** becomes the final differential diagnosis, DD_{EPS}. Since DD_{EPS} is initially null, count must be considered to avoid obtaining a null at all times (since $\forall\ X, X \cap \emptyset = \emptyset$). For the first while loop (first set of symptoms,), DD_{EPS} is $CDD_{nsymptoms}$ (11), and in subsequent loops, an intersection with $CDD_{nsymptoms}$. At the end of computations with the available set of symptoms, the WM is reset and computations continue with any new set of symptoms (13). The current DD_{EPS} is modified to include the PDs generated by the new set. The process continues until no new symptom is introduced to the problem at hand.

Note that all symptoms are accounted for in the final diagnosis. This is because the approach being used assumes the derivation of a set of disorders each of which explains all the symptoms. Reggia et al. (1983) agrees that if any symptom is not accounted for in the diagnosis, then we would arrive at an empty set. This is because if the disorders

in DD_{EPS} ($\{\delta_1, \delta_2, \ldots \delta_n\}$) are absent from the PD (say $\{\varepsilon_1, \varepsilon_2, \ldots \varepsilon_m\}$) for an unexplained symptom, then the intersection of the two sets would be Ø. The result is that each disorder in the final DD_{EPS} covers all the symptoms involved in the entire case being considered.

The processing done above is facilitated by the nature of the rule-organization in the Knowledge-Base. Each rule represents an IF ... THEN relationship of the Lisp-form:

(*rule-id* ((*precondition*$_1$) (*precondition*$_2$) ... (*precondition*$_n$))
(*rule-conclusion*)
(*resultant-disorders*))

Rule-id uniquely identifies each rule. Each precondition is of the form:

(*operator*, *keyword*, *keyword-value*),

where *operator* is a logical operator, *keyword* is a variable name (of a basic symptom or some intermediate conclusion), and *keyword-value* is the value being tested for *keyword* in the precondition. For example, (equal a 10) implies that the variable 'a' is expected to have the value '10' for that specific precondition to hold in that specific rule. The preconditions are implicitly recognized as the IF part of the rule (or antecedent) , while the *rule-conclusion* is the THEN part. *Resultant-disorders* is the set of disorders covered by the rule-conclusion, and is manually compiled from the DSM-III-R graph represented by the Knowledge-Base. We have included them in each rule as a check, to verify that the results obtained from our algorithms are accurate; we do not use them as computed solutions since they were compiled manually.

To obtain the overall antecedent, the preconditions are ANDed together in each rule. Any sets of preconditions involved in OR relationships are split into multiple rules. In each rule therefore, the relationship between preconditions is an AND. Rules which

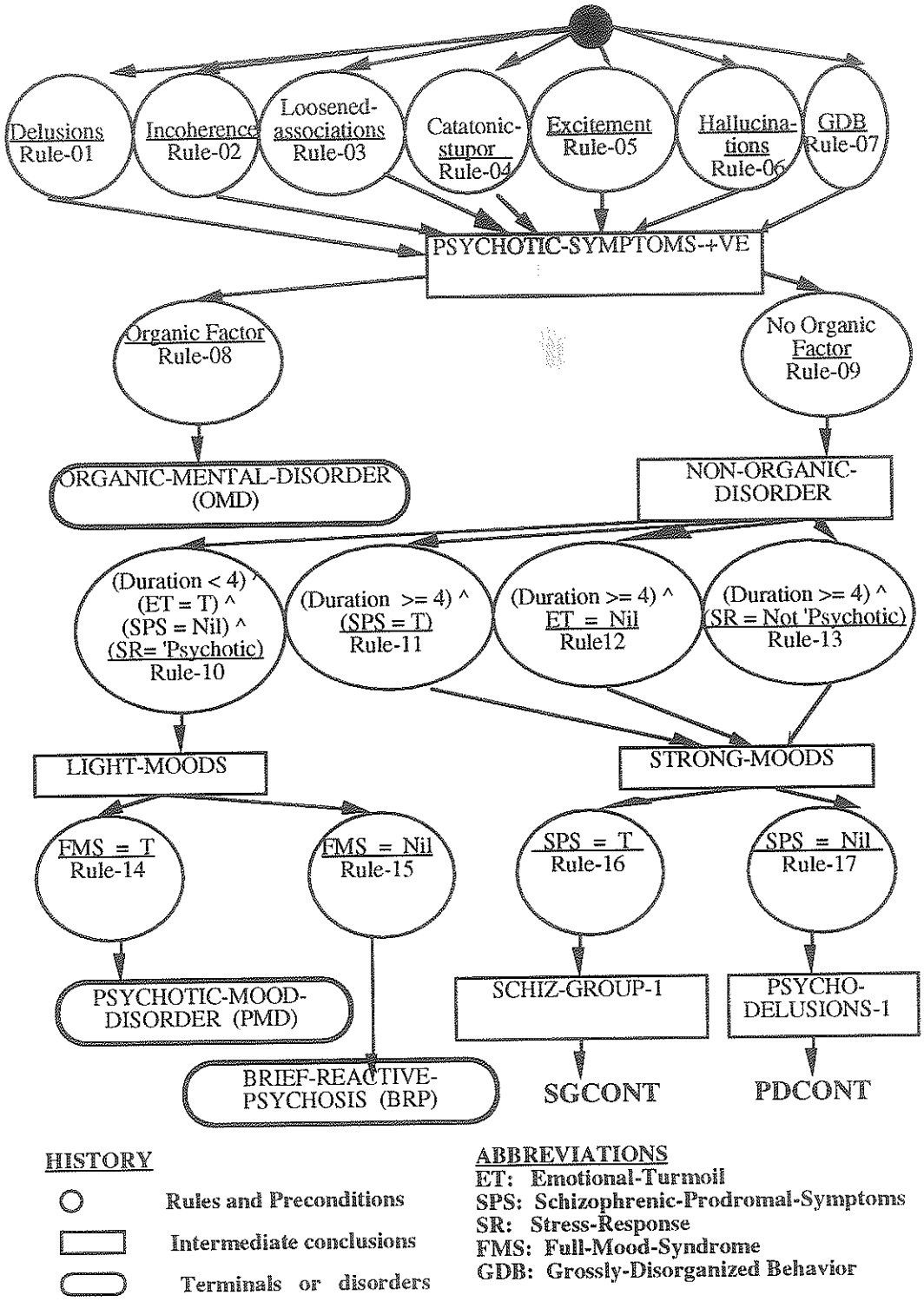


Figure 6.2. Top-level of hierarchical Knowledge-Base structure

resemble one another and differ only minimally, suggest an original OR relationship resulting in separation of the OR'ed preconditions. Maintenance of an AND-only relationship among preconditions was done to facilitate readability of rules, by allowing the view, in each rule, of a single group of preconditions associated with specific disorders. This separation also promotes consistency of rule structure, which feature promotes smoothness of implementation programs.

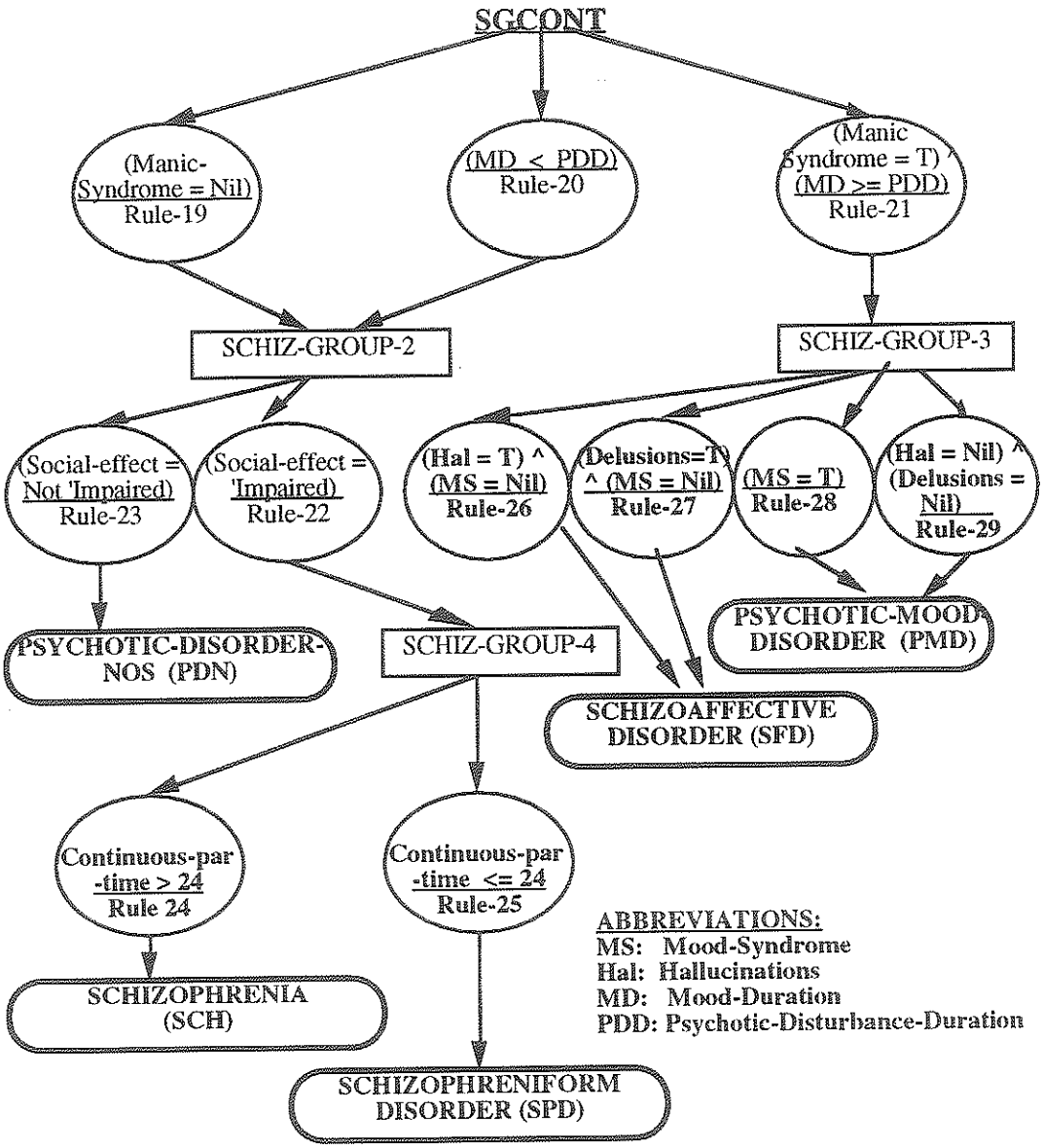
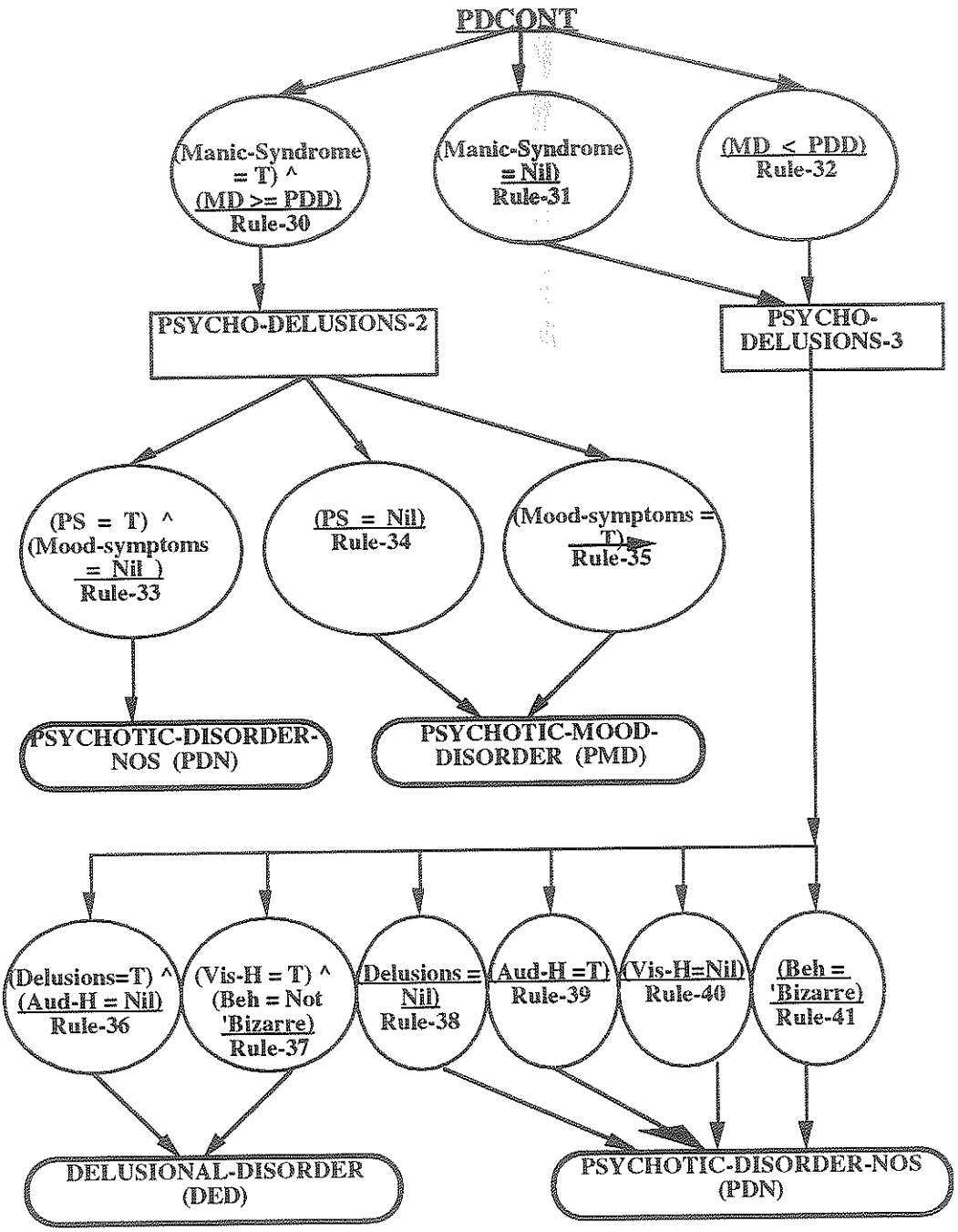


Figure 6.3. Continuation of rule-hierarchy, from SGCONT



ABBREVIATIONS:
MD: Mood-Duration
PDD: Psychotic-Disturbance-Duration
PS: Psychotic-Symptoms
Vis-H: Visual-Hallucinations
Aud-H: Auditory-Hallucinations
Beh: Behavior

Figure 6.4. Continuation of rule-hierarchy, from PDCONT

Therefore, given the modified rule-set presented in Appendix D, we obtain rule-graphs or networks as shown in Figures 6.2 to 6.4.

Algorithm 6.2: The rule-chainer

Input: new-prec: current precondition under search
searchbase: subset of $\mathcal{KB}$ being traced

Process: Forward-chain through searchbase to identify rules involving current symptom
Do union of disorders at end of chains

Variable names:
PD: Partial diagnosis (for a symptom)
i, j: loop variables
RC: rule-specific conclusion (disorder or an intermediate conclusion)
RPREC: rule-specific preconditions
new-prec: current precondition under search
searchbase: subset of $\mathcal{KB}$ being traced
branch-pt: a point of branch among rules in searchbase
branch-rule: the rule at which a branch is made
branch-prec: precondition for making next chain connection
Mark: indicates whether all of a rule's descendants have all been visited
σ: a symptom in $\mathcal{S}$
τ: a rule in $\mathcal{KB}$

Output: PD: symptom-specific partial diagnosis

```
procedure *chain-rules*(new-prec, searchbase)
   begin
     for each rule τj ∈ searchbase do                                   (1)
      if Mark_τj := 0 then      {go to next rule if current rule is marked}   (2)
         begin
         if new-prec ∈ RPREC_τj then                                    (3)
            begin
            branch-pt := branch-pt + 1                                  (4)
            branch-rule_branch-pt := τj
            branch-prec := RC_τj
            searchbase := searchbase - {τ1 ∪ τ2 ∪ . . . ∪ τj}
            if RC_τj ∈ Disorder-set then                                (5)
               begin
                  PD_σi := PD_σi ∪ RC_τj                                (6)
                  Mark_τj := 1                                          (6b)
```

```
            *backtrack*                                                    (7)
        end; {then}
        else begin
            new-prec := RC_τj                                              (8)
            *chain-rules*(new-prec, searchbase)                            (9)
        end; {if}
    end; {if new-prec}
  end; {if Mark_τj}
end; {for τj}
*backtrack*          {end of searchbase, or 'dead-end'}                    (10)
end; {*chain-rules*}
```

Algorithm 6.2. is a forward-chaining procedure which performs an exhaustive depth-first search of the searchbase under consideration, with a view to finding the disorders associated with the current symptom, σ_i. The reason for the forward-chaining is because the Knowledge-base is a multi-level hierarchy in which one rule's conclusion is involved in another rule's antecedent, and so on, until some rule at the end of a chain concludes an actual mental disorder. All rules concluding non-disorders are regarded as intermediate-conclusion rules. ***Chain-rules*** uses the procedure named ***backtrack*** (Algorithm 6.3), which backtracks to search unvisited rules. At the start of ***chain-rules*** for each symptom (new-prec), searchbase is equivalent to $\mathcal{KB}$, all of whose rules are marked zero (unvisited) for that symptom. The Algorithm performs a loop through searchbase (1) and aims for unmarked rules (2). First, it looks for the first rule in which new-prec is true (that is, a rule whose preconditions include new-prec) (3). If such a rule τ_j is found, it becomes the next branch-rule, with branch-pt incremented by 1 (4). The rule's conclusion (RC) becomes the branch-prec, which will be used later to chain the next logical point in the searchbase. Searchbase is then updated to become the rest of searchbase from τ_j. Next, the rule's conclusion is examined to determine whether it is some final disorder or not. If it is a disorder (5), that means we have reached the end of the current chain, and we update PD_{σ_i} to include the specific disorder, mark the current rule as visited, and then backtrack to the preceding branch

point (6) - (7). The reason for using a union operation to update the symptom's PD is that any of the disorders in its PD can cover the specified symptom. If on the other hand RC_{τ_j} is not a disorder, then it is an intermediate conclusion, and it becomes the new precondition to be used to chain the next logical point in searchbase (8) -(9). (9) is one recursive call within the ***chain-rules*** procedure. At that point, we are calling the depth-first search with the current rule current new-prec at the root. Steps (4) - (9) are consequent to step (3). At the end of the chain through the current searchbase, the procedure performs a backtrack to the preceding branch-point, to examine unvisited rules (10). There are two conditions under which ***backtrack*** is called: when the current searchbase has ended, or a disorder-concluding rule is reached. In all cases, the procedure understands that the current chain has terminated, and ***backtrack*** is called.

Algorithm 6.3: The backtrack controller

Input: branch-pt: a point of branch among rules in searchbase
branch-rule: the rule at which a branch is made
branch-prec: precondition for making next chain connection

Process: Determine if backtrack is possible
Determine most recent branch point
Set backtrack variables and activate ***chain-rules***

Variable names: i: loop variable
new-prec: current precondition under search
searchbase: subset of $\mathcal{KB}$ being traced
branch-pt: a point of branch among rules in searchbase
branch-rule: the rule at which a branch is made
branch-prec: precondition for making next chain connection
cbr: current branch rule
Mark: indicates whether all of a rule's descendants have all been visited
σ: a symptom in $\mathcal{S}$
τ: a rule in $\mathcal{KB}$

Output: nil

```
procedure *backtrack*( )
   begin
      cbr := branch-rule_branch-pt      {current branch rule}                (1)
      Mark_cbr := 1                     {mark as completely traversed}       (2)
      branch-pt := branch-pt - 1        {backtrack to preceding branch-point} (3)
```

```
if branch-pt = -1 then                                              (4)
   return-to compute-eps-diag
                                  {indicates no new pds for current symptom}
   else if branch-pt = 0 then                                       (5)
      begin                       {find new rules involving original symptom}
         new-prec := σ_i
         search-base := KB - {τ_1 ∪ τ_2 ∪ . . . ∪ cbr}
         *chain-rules*(new-prec, searchbase)     {chain again}
      end; {then}
    else begin                    {that is branch-pt is neither 0 nor -1}   (6)
         new-prec := branch-prec_branch-pt
         searchbase := KB - {τ_1 ∪ τ_2 ∪ . . . ∪ branch-rule_branch-pt}
         *chain-rules*(new-prec, searchbase)
    end; {if branch-pt = 0}
    end; {if branch-pt = -1}
end; {*backtrack*}
```

Algorithm 6.3 is the backtrack algorithm, which allows accurate backtracking in the chaining procedure. Cbr is the current rule from which the backtrack is to be made (1). First, this rule is marked *visited* (2). This step may appear to be a repeat of step (6b) of Algorithm 6.2., but that is not so because the latter is only one of the cases under which backtracking is performed, as discussed earlier. Under the end-of-searchbase condition, the rule to be marked is not the current rule, but the latest branch rule. In the case of (6b) of Algorithm 6.3., the latest branch point happens to be the current rule also. This is not so in the former case, and it is important to mark the current branch rule as *visited* before backtracking to the preceding branch point. A marked branch point tells the chaining procedure that all of the logical descendants of the current rule have been visited, and it would be computationally redundant to re-visit them. For the actual backtrack process, first branch-pt is decremented to point to the preceding branch-rule (3). Three classes of values may be obtained from this step. If branch point is anything other than zero or -1, new-prec is set to the branch precondition of the latest branch point, while searchbase is set to all rules from and excluding the latest branch-rule, after which ***chain-rules*** is called (6). If branch-pt is zero, that means the chain resulting

from the current start-point for the current symptom, has ended. The chainer must now look for a new start-point at which the current symptom also holds. This is necessitated by the AND-only structure of our Knowledge-base preconditions, the result of which a particular symptom may occur in more than one rule. In this backtrack case, therefore, new-prec is reset to the current symptom, and searchbase to all rules from and excluding branch-rule$_1$ (since branch-pt was 1 before decrement). Then ***chain-rules*** is called (5). The last possible value for branch-pt is -1, which means that a backtrack is being performed immediately after a backtrack from (branch-pt = 0). This implies that in the previous backtrack, no new start-point was found for which σ_i holds. The chainer must therefore exit to repeat the chaining procedure for the next symptom (4).

Algorithm 6.4: The final differential diagnosis recursion

Input: PDs: the partial diagnosis for all symptoms

Process: Recursively intersect current differential diagnoses
(*similar to the mathematical **factorial** recursion*)

Variables: i: loop variable
σ: a symptom in $\mathcal{S}$

Output: CDD: current differential diagnosis after considering current symptom

```
procedure loop-differential(nsymptoms)
  begin
    loop-differential(1) := PD_σ1                                   (1)
    for i = 2 to nsymptoms do
      CDD_i := PD_σi ∩ loop-differential(i-1);                      (2)
  end; {loop-differential}
```

This algorithm recursively intersects the PD of the current symptom, with the result of the same procedure performed on preceding symptoms, to obtain CDD. This is done for all the symptoms in a single set, referring to steps (2) - (5) of Algorithm 6.1. The

updated diagnoses have been obtained by a set intersection operation of current partial diagnoses with the previous differential diagnosis. This approach has been applied because a patient would normally go to a physician with a list of interacting symptoms (for example, headache **and** fever **and** vomiting, **and** so on). Also, we assume that each of the disorders in the final diagnosis must cover **all** the symptoms in the case.

Example 6.1

The following example illustrates the functions described in Algorithms 6.1 to 6.4. The task is to compute a differential diagnosis given a set of symptoms. Suppose we have the following symptoms:

- (equal excitement T)
- (equal organic-factor nil)
- (= duration 1)
- (equal schizophrenic-prodromal-symptoms nil)
- (equal emotional-turmoil T)
- (equal stress-response 'psychotic)
- (equal full-mood-syndrome nil)

This is a complete set of symptoms leading to the single-disorder differential diagnosis BRIEF-REACTIVE-PSYCHOSIS (BRP) when traced on the DSM-III-R-based graph for the diagnosis of psychotic disorders (Appendix C). Note that we use arithmetic operators (= < >) only for numeric variables. We shall illustrate in this example how our algorithms arrive at the same differential diagnosis.

Starting with the symptom (excitement = T), Algorithm 6.1. (**compute-eps-diag**) identifies Rule-05 as the first rule in which the symptom holds. It then passes control to ***chain-rules*** (Algorithm 6.2) for the rule-chaining process. The rule-chaining can be better understood by observation of the graph from Rule-05, shown in Figure 6.5. Two major chains emanate from Rule-05: that ending at Rule-08, and that continuing from Rule-09. Rule-05, Rule-08 and Rule-09 constitute branch-points 1, 2 and 3, respectively.

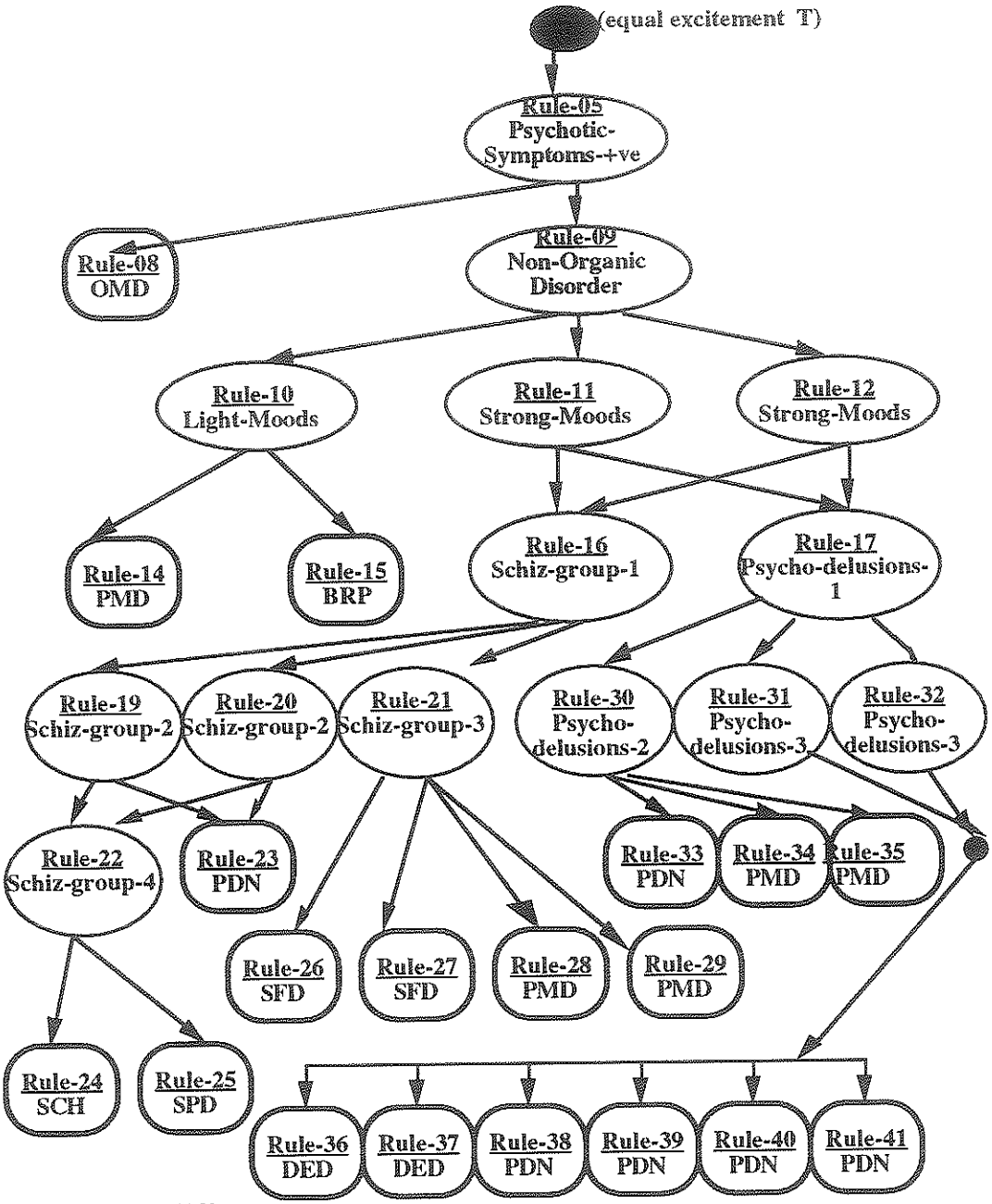


Figure 6.5. Rule-network for (equal excitement T)

After reaching Rule-08, ***chain-rules*** obtains a terminal disorder, and $PD_{(excitement = T)}$ becomes {Organic-Mental-Disorder}, or {OMD} for short. It then calls ***backtrack***, returning to the latest branch-rule (which is Rule-05). Continuing from Rule-09, ***chain-rules*** traverses to Rule-10, then to Rule-14 and Rule-15, updating $PD_{(excitement = T)}$ to {OMD, PMD, BRP}. It then **backtracks** to the latest branch-pt (Rule-09), and proceeds to Rule-11. This process continues, and summarizing the network from levels (top-down) and Left-to-Right, we obtain the overall partial diagnosis for (excitement =T) as

$PD_{(excitement = T)}$ = {OMD, PMD, BRP, SFD, SCH, SPD, PDN, DED},
or $PD_{(excitement = T)}$ = {BRP, DED, OMD, PDN, PMD, SCH, SFD, SPD},

sorted in alphabetical order. The same procedure is repeated for each other symptom.

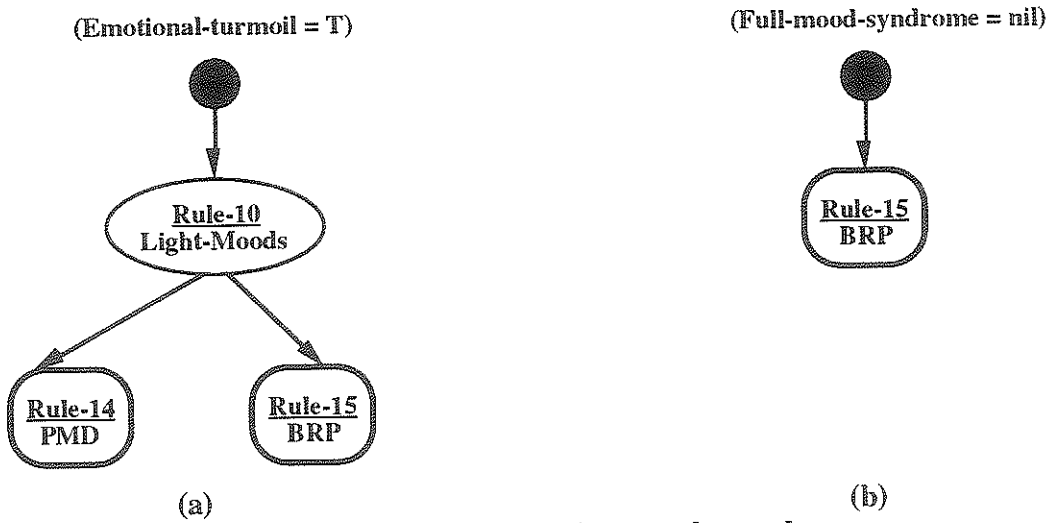


Figure 6.6. Low-density search graphs

The density of the search-graph for any symptom depends on the symptom's depth in the Knowledge-Base hierarchy of Figures 6.2 - 6.4, as well as the depths of the terminal nodes associated with the symptom. For example, Figure 6.6. shows the overall search-graphs for (emotional-turmoil = T) and (full-mood-syndrome = nil). Of particular note is Figure 6.6b, which contains only one node, which happens to be a terminal. This

graph would constrain the entire differential-diagnosis procedure to obtain a single disorder, since the differential diagnosis is obtained by intersecting all partial diagnoses. Thus, for a given symptom-set, a single-disorder-diagnosis is obtained **iff** there exists at least one single-node graph (multiple single-node graphs must necessarily contain identical nodes so that the entire search procedure does not produce a null set diagnosis). Otherwise, we expect the algorithm to produce one or more disorders in the final differential diagnosis.

Following the procedure described above, we obtain the following partial diagnoses for the given symptom-set:

$PD_{(\text{equal excitement T})}$ = {BRP, DED, OMD, PDN, PMD, SCH, SFD, SPD}
$PD_{(\text{equal organic-factor nil})}$ = {BRP, DED, PDN, PMD, SCH, SFD, SPD}
$PD_{(= \text{duration 1})}$ = {BRP, PDN, PMD, SCH, SFD, SPD}
$PD_{(\text{equal schizophrenic-prodromal-symptoms nil})}$ = {BRP, DED, PDN, PMD}
$PD_{(\text{equal emotional-turmoil T})}$ = {BRP, PMD}
$PD_{(\text{equal stress-response 'psychotic})}$ = {BRP, PMD}
$PD_{(\text{equal full-mood-syndrome nil})}$ = {BRP}

These are all obtained by ***chain-rules*** and ***backtrack***, and can be verified by observing the DSM-III-R graph of Appendix C. On return of control to **compute-eps-diag**, the last algorithm (**loop-differential**) is called for the final DD_{EPS}. **Loop-differential** then obtains values as follows:

$\textbf{loop-differential}_1$ = {BRP, DED, OMD, PDN, PMD, SCH, SFD, SPD}
$\textbf{loop-differential}_2$ = $PD_{(\text{organic-factor = nil})} \cap \textbf{loop-differential}_1$
= {BRP, DED, PDN, PMD, SCH, SFD, SPD}
$\cap$ {BRP, DED, OMD, PDN, PMD, SCH, SFD, SPD}
= {BRP, DED, PDN, PMD, SCH, SFD, SPD}

•
•
•

$\textbf{loop-differential}_7$ = $PD_{(\text{full-mood-syndrome = nil})} \cap \textbf{loop-differential}_6$
= {BRP} $\cap$ {BRP, PMD}
= {BRP}

On return to **compute-eps-diag**, DD_{EPS} is set to {BRP}, and the algorithm awaits another symptom set, in whose absence the process is terminated.

6.7 Knowledge-finding: Computing alternative paths to same solution

There are theoretically four possible situations involved in a student's response or problem-solving results:

Correct Response + Correct Reasoning	(CRCR)
Correct Response + Incorrect Reasoning	(CRIR)
Incorrect Response + Incorrect Reasoning	(IRIR)
Incorrect Reasoning + Correct Reasoning	(IRCR)

The last alternative is impossible because an incorrect response always comes as a result of some incorrect reasoning at some point (even if it is at the very last step) of the solution process.

The aim of effective modeling is to ensure that the student's response is both correct and complete, without putting any restrictions on solution strategy. We have therefore concentrated on designing a strategy-independent evaluation model, whereby the student is permitted to solve a problem in any desired way, as long as the result is correct and complete. This capability is made possible by the Knowledge-Finder, which computes all alternative paths to the same solution obtained by the Expert Problem Solver in Section 6.6 above. The alternative paths are then passed on to the Response Analyzer which would determine which candidate paths the student is applying in a solution. These paths are computed using the Knowledge-Base. This component of our design has also been implemented, and Algorithms 6.5 and 6.6 describe the procedural details.

Algorithm 6.5: The Knowledge-Finder's alternative paths

Input: $\mathcal{BK}$: set of rules (reverse knowledge-base)
DD_{EPS}: EPS's differential diagnosis

Process: Pass through $\mathcal{BK}$ to obtain alternative reasoning paths

Variables: DD: Differential diagnosis
BD: Base diagnosis
RG: Recurrent Group
nrules: number of rules in BK
RD: rule-specific differential diagnosis
RGN: rule-specific diagnosis group name
PPREC: Principal precondition in rule
i, j, m: loop variables
$\mathcal{SL}$: Search List (subset of BK)
PB: Path Buffer
INC-PATHS: an array of incomplete paths
INC-LISTS: an array of search lists for INC-PATHS
npaths: number of alternative paths
ipcnt: number of incomplete paths
ρ: a path in INC-PATHS
τ: a rule in $\mathcal{BK}$ or $\mathcal{SL}$
update-paths: path update procedure (see **Algorithm 6.3**)

Output: $\mathcal{SB}$: set of alternative strategies (paths) to DD_{EPS}

```
procedure compute-alternative-strategies( DD_EPS)
  begin
    npaths = ipcnt := 0                                  {initialize variables}        (1)
    SB = INC-PATHS = INC-LISTS = ∅
    for each rule τ_i ∈ BK do                                                          (2)
      begin
        if RD_τi ⊇ DD_EPS then                           {beginning of a major path}   (3)
          begin
            PB := τ_i                                    {initialize path buffer}      (4)
            BD := PPREC_τi
            RG := RGN_τi
            SL := BK - (τ_i ∪ τ_i+1 ∪ ... ∪ τ_nrules)                                  (5)
            update-paths                                 {call update procedure}       (6)
            while ∃ some new ρ_j's (j=1 to m), ⊆ INC-PATHS do                          (7)
              for each ρ_j do                                                          (8)
                begin
                  SL := INC-LISTS_ρj
                  PB := INC-PATHS_ρj
                  RG := RGN_τ1
                  BD := PPREC_τ1   {initialize BD to PPREC}                          (9)
                                   {of first rule in SL}
                  update-paths;                                                      (10)
              end; {for ρ_j}
```

```
            end; {while}
        end; {if}
    end; {for τi}
end {compute-alternative-strategies}
```

Algorithm 6.5. produces a set of paths or strategies ($\mathcal{SB}$) which lead to the same diagnosis obtained by the EPS in Algorithm 6.1. It takes the DD_{EPS} as input and performs a backward chain through the knowledge-base to obtain all possible paths to the same conclusion. This is an exhaustive listing which involves depth-first search and chronological backtracking at several points of the knowledge base. At every backtracking point, the current sub-path is saved into an incomplete-paths data structure, for future completion. This procedure is similar to the combined set of Algorithms 6.1 to 6.4, except that a different approach is taken since whole paths are saved rather than just the accumulation of final disorders done in preceding algorithms. Thus a node in the above graph may be visited more than once, in order to obtain an exhaustive set of paths leading to the EPS's final diagnosis.

At the start of the procedure, variables are initialized, including the Strategy-base ($\mathcal{SB}$) which is empty. INC-PATHS contains incomplete paths which will be completed by tracing the sub-knowledge-bases stored in INC-LISTS (1). We have reversed the Knowledge-Base ($\mathcal{KB}$) to obtain $\mathcal{BK}$, which allows backward tracing of $\mathcal{KB}$. Then each rule in $\mathcal{BK}$ is analyzed to determine whether its conclusion contains DD_{EPS} (2). Any rule concluding DD_{EPS} becomes a starting point (or root) of possibly several paths (3). If a rule, say τ_i does conclude DD_{EPS}, $\mathcal{BK}$ is split to be replaced by untraced portions of $\mathcal{BK}$ up to τ_i. This is the search-list, $\mathcal{SL}$ (5). The path buffer (PB) is initialized to start at τ_i, and the base diagnosis BD is initialized to τ_i's principal precondition $PPREC_{\tau_i}$ while the recurrent-group RG is initialized to the rule-group-name of τ_i, RGN_{τ_i} (4). BD is used to connect to the next point of the path, in that the

principal precondition of the current rule τ_i must have been concluded by some later (in $\mathcal{BK}$) rule τ_j. Once such τ_j is found such that $PPREC_{\tau_i} = RGN_{\tau_j}$, then we get rule τ_j connected as the next path point. RG is used to identify points of multiple path creation. If a rule-group-name occurs in more than one rule, then multiple paths are created to accommodate each occurrence. Each new occurrence (say at rule τ_m) is saved as the end of an incomplete path up to τ_m. Once these variables are initialized, the **update-paths** procedure (Algorithm 6.6) is called to complete the trace (6), after which incomplete paths are completed in (7) and (8). Incomplete paths are traced by reloading each incomplete path along with its search list SL, from INC-PATHS and INC-LISTS, respectively. Then BD and RG are initialized and the complete tracing is performed by **update-paths** (10). Further incomplete paths can result from this trace, so the incomplete-path tracing is a recursive process (7).

The result is a set of paths, some of which may have some rules in common, representing disorders repeated in several paths. This is expected, since each path is a correct route to the final EPS diagnosis, and each may involve all the symptoms used in the initial derivation of DD_{EPS}. Differences in paths may include order of conclusions, and other symptoms which may be assumed to arrive at the same conclusion. It is left to the Tutor (in conjunction with the Response Analyzer) to query to determine what assumptions the student is making, and what paths he/she may be following to arrive at some conclusion.

Algorithm 6.6: Update-paths procedure

Input: BD: Base Diagnosis
PB: Path Buffer
RG: Recurrent Group
$\mathcal{SL}$: Search List
$\mathcal{DVC}$: set of domain-specific derivable conclusions

Process: Connect new points to path

Create new subsidiary paths

Variables:
- nrules: number of rules in BK
- RD: rule-specific differential diagnosis
- RC: rule-specific conclusion
- PPREC: Principal precondition in rule
- j: loop variable
- INC-PATHS: an array of incomplete paths
- INC-LISTS: an array of search lists for INC-PATHS
- npaths: number of alternative paths
- ipcnt: number of incomplete paths
- ρ: a path in INC-PATHS
- τ: a rule in $\mathcal{BK}$ or $\mathcal{SL}$

Output: Updated INC-PATHS, INC-LISTS, $\mathcal{SB}$

```
procedure    update-paths( )
  begin
    for each rule τj ∈ SL do                                              (1)
      begin
        if RCτj = BD then                                                 (2)
          begin                                  {Connect new point}
            PB := PB ∪ τj
            if PPRECτj ∈ DVC                                              (3)
              then BD := PPRECτj                                          (4)
              else begin      {end of path}
                     npaths := npaths + 1
                     SBnpaths := reverse PB                               (5)
                     PB := PB - τj                                        (6)
                                            {make room for more end points}
                   end; {else}
          end {if}
        if RCτj = RG then                    {Create subsidiary path}     (7)
          begin
            ipcnt := ipcnt + 1
            INC-PATHSipcnt := PB ∪ τj                                     (8)
            INC-LISTSipcnt := SL - (τj+1 ∪ τj+2 ∪ . . . ∪ τnrules)        (9)
          end; {if}
      end; {for τj}
  end {update-paths}
```

Algorithm 6.6. performs the tracing of a path initialized in Algorithm 6.5, from which it receives BD (base diagnosis), RG (recurrent group), SL (search list) and the path buffer, PB. Essentially, $\mathcal{SL}$ is traced to look for BD and RG (1). If a rule τ_k is found concluding BD (2), τ_k becomes the next point in the current path, as described above (3)-(6). Rule τ_k is added to the current path buffer, and the path ends if τ_k's principal precondition (PPREC) is an indecomposable symptom (5)-(6), otherwise ($PPREC_{\tau_k}$) is a derivable conclusion, τ_k is simply the next point in the current path (4). If τ_k concludes RG, then a new incomplete-path is created and appropriate contents and search list saved into INC-PATHS and INC-LISTS, respectively (7)-(9).

Example 6.2

As an example, let us follow up on Example 6.1's DD_{EPS}, {BRIEF-REACTIVE-PSYCHOSIS}. The aim of the Knowledge-Finder is to compute all paths that could be followed to arrive at the same conclusion. To better visualize this process, Figure 6.7. shows a backward graph with the single node concluding {BRIEF-REACTIVE-PSYCHOSIS} (Rule-15). Circled numbers in Figure 6.7 represent rule-ids, and we have removed rule preconditions and conclusions for easier readability. Exhaustive reverse listing of this graph gives us the following 7 paths (listing only the rule-identifiers, for readability):

(Rule-01 Rule-09 Rule-10 Rule-15)
(Rule-02 Rule-09 Rule-10 Rule-15)
(Rule-03 Rule-09 Rule-10 Rule-15)
(Rule-04 Rule-09 Rule-10 Rule-15)
(Rule-05 Rule-09 Rule-10 Rule-15)
(Rule-06 Rule-09 Rule-10 Rule-15)
(Rule-07 Rule-09 Rule-10 Rule-15)

Including the preconditions and conclusions would give us, for example, the first path as the following (comments italicized):

(((equal delusions T), psychotic-symptoms-positive) *{ for Rule-01 }*

((equal psychotic-symptoms-positive T)^ (equal organic-factor nil)), non-organic-disorder) *{ for Rule-09}*

((equal non-organic-disorder T) ^ (<duration 4) ^ (equal schizophrenic-prodromal-symptoms nil) ^ (equal emotional-turmoil T) ^ (equal stress-response 'psychotic)), light-moods) *{ for Rule-10}*

((equal light-moods T) ^ (equal full-mood-syndrome nil)) brief-reactive-psychosis)) *{ for Rule-15}*

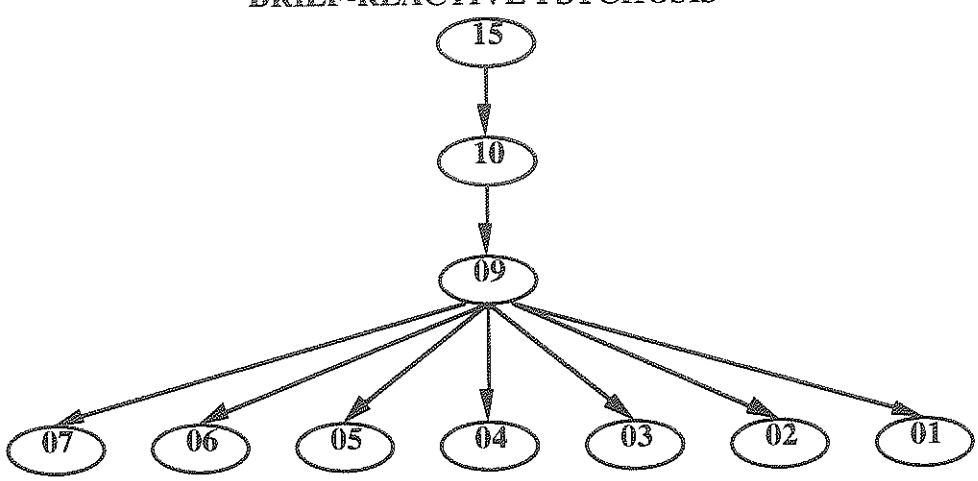


Figure 6.7. Knowledge-Finder's graph for Brief-Reactive-Psychosis

Notice that Rule-09 always branches off to Rules-01 to -07. We can therefore heuristically reduce the search procedure by letting the algorithm automatically chain Rules-01 to -07 anytime it reaches Rule-09. The same heuristic can be applied to any node branching off into the same set of multiple nodes. Notice also that the total number of paths in a graph can be computed as the sum of the product of the number of paths in each separately linked sub-graph:

$$\text{npaths} = \sum_{pr=1}^{k} pr, \text{ where } pr = \prod_{br=1}^{k} br, \text{ where br is the number of branches at each}$$

level of the sub-graph. For example, in Figure 6.7, there is only one separate sub-graph in the overall graph, with pr = (1 * 1 * 1 * 7) = 7 paths.

Applying a denser example, suppose that the EPS arrived at the {PSYCHOTIC-MOOD-DISORDER}, Figure 6.8. represents the graph whose top-level nodes are the five rules concluding {PSYCHOTIC-MOOD-DISORDER}.

This graph contains three separately-linked sub-graphs (that from Rule-14, that from Rules-34/35, and that from Rules-28/29).

This gives us

npaths = (1*1*1*7) + (2*1*1*3*1*7) + (2*1*1*3*1*7)
= 7 + 42 + 42
= 91

Exhaustive search and listing of this graph produces ninety-one paths.

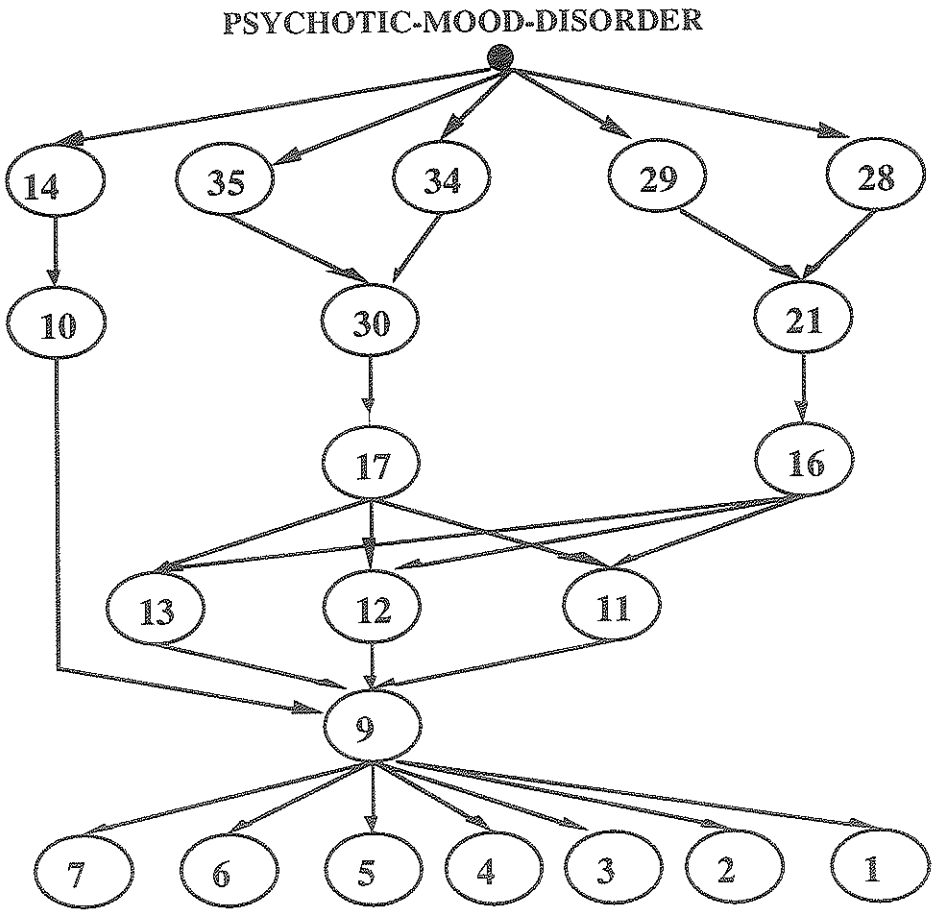


Figure 6.8: Graph of paths concluding Psychotic-Mood-Disorder

6.8 Response Analysis: Identification of student's beliefs and strategies

It is not expected that the student would always correctly apply any of the strategies or paths discovered by the Knowledge-Finder in Section 6.7 above. The Tutor expects intermingling, omissions, and extra inclusions, which lead to incorrect responses. However, by querying the student for justifications, alternative reasoning, and underlying assumptions, the tutoring model can determine which of the sequences the student is applying, albeit, skeletally. The task of uncovering the student's beliefs and strategies is performed by the Response Analyzer. Beliefs are identified using queries, while solution strategies are identified by computing the Knowledge Finder paths which contain the highest frequencies of the student's current beliefs. Algorithms 6.7 to 6.11 describe the details of how the Response Analyzer performs these functions. Unless otherwise specified, belief symbols refer to the basic belief (*antecedent* $\Rightarrow$ *conclusion*). We have left out belief identifiers, types and justifier sets (which will be included in an overall belief structure by the belief revision unit) for convenience. This also applies to the Deviation Finder's algorithms.

Algorithm 6.7: Response-Analyzer's beliefs

Input: Tutor-Student Interaction
$\mathcal{SB}$: Strategy-base (set of alternative strategies)

Process: Identify student beliefs
Identify student strategies

Variables:
A, C, D, E, F, SR: Interaction and belief variables
QJ, QN, QA: Query type abbreviations
β: A single belief $\{\beta \in \mathcal{B}\}$
bcnt: Number of student beliefs per strategy
i, j: loop variables
φ: a single strategy $\{\varphi \in \mathcal{SB}\}$
θ: threshold for determination of candidate strategies

Output: $\mathcal{B}$: Set of Beliefs
$\mathcal{CS}$: Set of candidate strategies $\{\mathcal{CS} \subseteq \mathcal{SB}\}$

```
procedure     identify-beliefs( )
   begin
      generate random number                                                    (1)
      case    {Phase One:  Identify beliefs using randomly generated queries}   (2)
         1:  query-for-justification(A, SR)                                     (3)

         2:  query-for-negation(C, SR)                                          (4)

         3:  (query-for-assumption(A, SR)                                       (5)

      end; {case}
      for each compound belief do                  {say ((A ^ C) ⇒ D)}          (6)
        begin                                      {separate into single units}
         if connector is AND  then  create concurrent set                       (7)
                                                   {e.g. ((A ⇒ D)(C ⇒ D))}
         if connector is OR  then  create multiple sets                         (8)
                                                   {e.g. (A ⇒ D), (C ⇒ D)}
        end; {for}
   end {identify-beliefs}
```

Algorithm 6.7 represents the first major task of the Response Analyzer: identification of the student's beliefs. This is done by getting the Tutor to present the student with a series of queries of different types (see Algorithms 6.8 - 6.10). First, a random number is generated to identify what type of query is to be posed to the student (1). Then based on the random number generated, queries are posed in (3) - (5). "A" in (3) and (5) signifies the most recent student response, which is the subject of the queries. "SR" signifies an expected (new) response which the student will give after a query has been posed. "C" in (4) is an alternative value for the keyword of the Tutor's latest question, which is generated to test the student's reasoning about alternative situations.

After obtaining beliefs, the Response Analyzer identifies those beliefs that contain multiple antecedents (compound beliefs), and breaks them up into single-antecedent beliefs (6). For instance, multiple beliefs involving an AND operator are split into multiple **concurrent** single-antecedent beliefs (7), while OR-ed beliefs are split into multiple disjoint single-antecedent beliefs (8). The reason for this split is to facilitate the

labeling of beliefs in traces which will be done by the Deviation Finder in Algorithms 6.12 and 6.15.

Algorithm 6.8: Query-for-justification

Input: A: Student input

Process: Ask question
Obtain student response, SR

Output: β_{QJ}: a belief

```
procedure query-for-justification(A, SR)
   begin
      print ('How did you arrive at' A '?')          (1)
      input SR                                       (2)
      βQJ := (SR ⇒ A)                                (3)
   end {query-for-justification}
```

Algorithm 6.8 obtains the student's response to a query based on his or her last input. A **query-for-justification** asks for the student's reason (2) behind an input A (1). After the student offers a response (SR), the Response Analyzer generates a new belief signifying that the student believes that SR is a justification for the previous response, "A" (3)

Algorithm 6.9: Query-for-negation

Input: A: Student input
K: keyword of previous Tutor question

Process: Ask question
Obtain student response, SR
Perform **query-for-justification** on SR

Variables:
D, Y, SR: belief parts
range-start: beginning value for a variable's range of values
range-end: last value for a variable's range of values
K: a keyword, signifying a variable

Output: β_{QN}: a belief

β_{QJ}: a belief

```
procedure query-for-negation(Y, SR)
   begin
      generate(Y | range-start_K ≤ Y ≤ range-end_K)                    (1)
      print (' Suppose ', K, ' had been ', Y, '?')                     (2)
      input SR                                                         (3)
      β_QN := (Y ⇒ SR)                                                 (4)
      query-for-justification(SR, D)                                   (5)
      β_QJ := (D ⇒ SR)
   end {query-for-negation}
```

This algorithm represents a query for negation or alternative reasoning. An alternative query-startpoint (2) is presented to the student, to determine how the latter's response would be affected by viewing the current problem from a different perspective (3). The alternative values must fall within the range of affected keywords (1). Note that **query-for-negation** involves **query-for-justification**, since the student is required to justify the reasoning behind responses given to negation.

Algorithm 6.10: Query-for-assumption

Input: A: Student input

Process: Ask question
Obtain assumptions (α_1 . . α_{cnt})
Perform **query-for-justification** on (α_1 . . α_{cnt})

Output: β_{QA}: a belief
β_{QJ}'s: beliefs

```
procedure query-for-assumption(A, SR)
   begin
      print (' What assumptions did you make to conclude ', A, '?')    (1)
      i := 1
      input α_1                                                        (2)
      while α_i <> null do                                             (3)
         begin
            i := i+1
```

```
            input α_i
        end; {while}
    cnt := i - 1                                              (4)
    SR := (α_1 ^ α_2 ^ ... ^ α_cnt)                           (5)
    β_QA := (SR ⇒ A)                                          (6)
    for k:= 1 to cnt do                                       (7)
      begin
            query-for-justification(α_k , f_k)                (8)
            β_QJ_k := (f_k ⇒ α_k)                             (9)
      end; {for k}
end {query-for-assumption}
```

Algorithm 6.10 collects the assumptions which guided the student in obtaining a previous response, A (1). The first assumption α_1 (2) is collected to initialize the student's response, SR, and the remaining α's are collected in (3). The reason for this is to enable execution of the **while** loop in (3). The final SR is the AND of all the α's (5), since all the assumptions collectively led the student to the initial response A. Again, **query-for-justification** is needed to obtain further student beliefs explaining reasons behind assumptions.

Algorithm 6.11: Response-Analyzer's strategies

Input: $\mathcal{SB}$: Strategy-base (set of alternative strategies)
$\mathcal{B}$: Set of Beliefs

Process: Identify student beliefs
Identify student strategies

Variables: β: A single belief $\{\beta \in \mathcal{B}\}$
bcnt: Number of student beliefs per strategy
i, j: loop variables
φ: a single strategy $\{\varphi \in \mathcal{SB}\}$
θ: threshold for determination of candidate strategies

Output: $\mathcal{CS}$: Set of candidate strategies $\{\mathcal{CS} \subseteq \mathcal{SB}\}$

procedure identify-strategies( )

```
begin                                   {Phase Two:  Identify candidate strategies}
   for each strategy φ_i ∈ SS do                                                  (1)
     begin
        bcnt_φi := 0                    {belief count}                          (2)
        for each β_j ∈ B do                                                     (3)
           if β_j ∈ φ_i then bcnt_φi := bcnt_φi + 1;                            (4)
     end; {for φ_i}
   sort SS in descending order of bcnt                                          (5)
   CS := { all φ_i | bcnt_φi ≥ θ}       {θ is some predetermined threshold}     (6)
end {compute-beliefs-strategies}
```

Algorithm 6.11 identifies the strategies in $\mathcal{SS}$ being applied by the student, using the set of beliefs obtained in Algorithm 6.7. Each strategy has an associated belief count ($bcnt_{\varphi_i}$) which represents the number of student-beliefs contained in the specific strategy. bcnt is computed for all strategies in $\mathcal{SS}$ (1)-(4), after which strategies are sorted in descending order of bcnt (5). The sort algorithm here depends largely on the number of strategies, ns. For large ns, (say ≥ 1000), we may use the **quicksort** algorithm which is of $O((ns)^2)$. For small ns, we can get away with less complicated algorithms like **replacement-sort** or **insertion-sort** which are of $O((ns)^3)$. The candidate strategies (CS) are those with bcnt ≥ a predetermined threshold, θ (6). We may assume that if the number of student beliefs is nb, a reasonable θ would be $\frac{2}{3}$nb.

Note that obtaining a notion of correctness of the ordering of student beliefs within a strategy is not performed by the Response Analyzer, but by the Deviation Finder in the next modeling phase. The Response Analyzer is mainly concerned with obtaining student beliefs, whether correct or incorrect. For instance, if a correct belief is, say ((manic-syndrome = T) ⇒ delusional-disorder), while the student holds an opposite belief, ((manic-syndrome = nil) ⇒ delusional-disorder), all the Response Analyzer does is to detect that the student does hold that belief, leaving the actual correctness labeling to the Deviation Finder.

Example 6.3

Suppose the student receives the same set of symptoms presented in Example 6.1., and obtains a different diagnosis, say {PSYCHOTIC-MOOD-DISORDER}. The Knowledge-Finder then obtains the strategies shown in Example 6.2, with all details included as follows:

φ_1: (((equal delusions T), psychotic-symptoms-positive) *{ for Rule-01}*
((equal psychotic-symptoms-positive T)^ (equal organic-factor nil)), non-organic-disorder) *{ for Rule-09}*
((equal non-organic-disorder T) ^ (<duration 4) ^ (equal schizophrenic-prodromal-symptoms nil) ^ (equal emotional-turmoil T) ^ (equal stress-response 'psychotic)), light-moods) *{ for Rule-10}*
((equal light-moods T) ^ (equal full-mood-syndrome nil)) brief-reactive-psychosis))
{ *for Rule-15}*

φ_2: (((equal incoherence T), psychotic-symptoms-positive) *{ for Rule-02}*
((equal psychotic-symptoms-positive T)^ (equal organic-factor nil)), non-organic-disorder) *{ for Rule-09}*
((equal non-organic-disorder T) ^ (<duration 4) ^ (equal schizophrenic-prodromal-symptoms nil) ^ (equal emotional-turmoil T) ^ (equal stress-response 'psychotic)), light-moods) *{ for Rule-10}*
((equal light-moods T) ^ (equal full-mood-syndrome nil)) brief-reactive-psychosis))
{*for Rule-15}*

φ_3: (((equal loosened-associations T), psychotic-symptoms-positive) *{ for Rule-03}*
((equal psychotic-symptoms-positive T)^ (equal organic-factor nil)), non-organic-disorder) *{ for Rule-09}*
((equal non-organic-disorder T) ^ (<duration 4) ^ (equal schizophrenic-prodromal-symptoms nil) ^ (equal emotional-turmoil T) ^ (equal stress-response 'psychotic)), light-moods) *{ for Rule-10}*
((equal light-moods T) ^ (equal full-mood-syndrome nil)) brief-reactive-psychosis))
{ *for Rule-15}*

φ_4: (((equal catatonic-stupor T), psychotic-symptoms-positive) *{ for Rule-04}*
((equal psychotic-symptoms-positive T)^ (equal organic-factor nil)), non-organic-disorder) *{ for Rule-09}*
((equal non-organic-disorder T) ^ (<duration 4) ^ (equal schizophrenic-prodromal-symptoms nil) ^ (equal emotional-turmoil T) ^ (equal stress-response 'psychotic)), light-moods) *{ for Rule-10}*
((equal light-moods T) ^ (equal full-mood-syndrome nil)) brief-reactive-psychosis))
{ *for Rule-15}*

φ_5: (((equal excitement T), psychotic-symptoms-positive) *{ for Rule-05}*
((equal psychotic-symptoms-positive T)^ (equal organic-factor nil)), non-organic-disorder) *{ for Rule-09}*
((equal non-organic-disorder T) ^ (<duration 4) ^ (equal schizophrenic-prodromal-symptoms nil) ^ (equal emotional-turmoil T) ^ (equal stress-response 'psychotic)), light-moods) *{ for Rule-10}*

((equal light-moods T) ^ (equal full-mood-syndrome nil)) brief-reactive-psychosis))
{ for Rule-15}

φ_6: (((equal hallucinations T), psychotic-symptoms-positive) *{ for Rule-06}*
((equal psychotic-symptoms-positive T)^ (equal organic-factor nil)), non-organic-disorder) *{ for Rule-09}*
((equal non-organic-disorder T) ^ (<duration 4) ^ (equal schizophrenic-prodromal-symptoms nil) ^ (equal emotional-turmoil T) ^ (equal stress-response 'psychotic)), light-moods) *{ for Rule-10}*
((equal light-moods T) ^ (equal full-mood-syndrome nil)) brief-reactive-psychosis))
{ for Rule-15}

φ_7: (((equal behavior 'grossly-disorganized), psychotic-symptoms-positive)
{ for Rule-07}
((equal psychotic-symptoms-positive T)^ (equal organic-factor nil)), non-organic-disorder) *{ for Rule-09}*
((equal non-organic-disorder T) ^ (< duration 4) ^ (equal schizophrenic-prodromal-symptoms nil) ^ (equal emotional-turmoil T) ^ (equal stress-response 'psychotic)), light-moods) *{ for Rule-10}*
((equal light-moods T) ^ (equal full-mood-syndrome nil)) brief-reactive-psychosis))
{ for Rule-15}

It is now the Response Analyzer's task to query to determine which basic strategy the student applied to his/her solution. Let us assume the following conversation occurs ('T' below represents 'Tutor'. Our reason for showing queries as coming from the Tutor is because the Tutor is the direct interface with the student. The Response Analyzer is therefore presenting the queries with the Tutor's assistance. Once again, we have assumed a highly structured communication language for the student and the Tutor.).

T: How did you arrive at {PSYCHOTIC-MOOD-DISORDER}?
S: Because (equal stress-response 'psychotic) and (< duration 4)
T: What assumptions did you make in this problem?
S: (equal incoherence T)
T: Why did you assume that?
S: It suggests {psychotic-symptoms-positive}
T: How did you apply the other symptoms in this problem?
S: (equal excitement T) suggests {psychotic-symptoms-positive}
{equal organic-factor nil) suggests {non-organic-disorder}

From this interaction, the Response Analyzer obtains the beliefs:

β_1: (equal stress-response 'psychotic) $\Rightarrow$ psychotic-mood-disorder

β_2: (< duration 4) $\Rightarrow$ psychotic-mood-disorder

β_3: (equal incoherence T) $\Rightarrow$ psychotic-symptoms-positive

β_4: (equal excitement T) $\Rightarrow$ psychotic-symptoms-positive

β_5: (equal organic-factor nil) $\Rightarrow$ non-organic-disorder

Beliefs β_1 and β_2 happen to be absent from the strategies φ_1 - φ_7, so they are not computed into any of the strategies' belief counts. However, β_3 is present in φ_2, β_4 in φ_5 and β_5 in all of φ_1-φ_7. The Response Analyzer would therefore arrive at a *CS* consisting of φ_2 and φ_5, if the bcnt threshold (θ) is 2, since these two strategies have the highest frequencies of the student's current beliefs. This is a small-range example, but the same technique is followed in larger cases.

6.9 Deviation-Finding: Labeling beliefs and identifying misconceptions

The reasoning-tracing or query process performed by the Response Analyzer in Section 6.8 above is undertaken in order for the Tutor to decipher answers to the following questions:

- What is the student thinking?
- What does the student understand?
- What does the student not understand?
- What more information should the student have applied?
- What are the student's underlying misconceptions?

These questions are difficult cognitive problems bordering the process of modeling a student's beliefs. The first question is answered by the Response Analyzer, which draws out student beliefs. The last three questions are answered by the Deviation Finder, which aims at labeling student beliefs as correct (answer to second question) or incorrect (answer to third question), identify missing beliefs (answer to the fourth question), and also identifies the misconceptions underlying a student's incorrect response (answer to last question).

Our deviation-finding approach is done using the knowledge-base, rather than a mal-rule-set. As explained in Chapter 5, this approach allows for more accurate modeling,

since the student's misconceptions are outlined in terms of correct knowledge rather than a previously represented incorrect template.

Although we check correctness using the knowledge-base rules, ours is not an overlay model, for the following reasons:

a) The student's response is not matched side-by-side with the EPS's solutions. Thus the student is allowed to solve a problem in any comfortable manner, without fear of penalization.

b) The model is a reason-tracing model, after the correctness and completeness of a student's solution.

c) From b), correctness of a student's solution is not based on the final response, as is the case in a typical overlay model. Intermediate steps are checked for their correctness and completeness, and interactions with the student are performed to ensure that concepts are clearly understood by the student.

Secondly, this is not a Buggy model, for the following reasons:

a) The student is not interrupted during the problem-solving process but is allowed an independent hand until s/he specifies completion, at which point the evaluator is activated. This is therefore a non-intrusive model

b) There does not exist a set of mal-rules that attempt to explain the student's behavior. We strongly believe that for every piece of knowledge, there is a large number of misconceptions. This is because each student has his/her own unique misconceptions. It is therefore difficult to produce an exhaustive mal-rule-base. However, since every misconception can be identified as a deviation from correct facts, our model attempts to explain a student's errors in terms of deviations from the knowledge-base rules. This

was the approach employed by GUIDON, which evaluated the student based on MYCIN rules. Several differences however exist between our model and GUIDON's, including the following:

a) GUIDON used a strictly overlay model, while ours is strategy-independent.

b) GUIDON could not perform constructive evaluation in the sense of understanding student misconceptions and offering explanations for them. We however achieve this feature by explaining misconceptions as deviations from rules (explanation of misconceptions is a buggy property).

c) GUIDON was an immediate *answer-provider* since it supplied correct responses to the student without giving enough room for the student to figure out the problem. We however take a Socratic approach, giving hints and prods, until the student figures out the problem without being told explicitly by the tutor. The Socratic approach makes for better understanding, since the student is actively involved in the solution and reasoning process.

The fact that a student's intermediate beliefs are analyzed is illustrated by the fact that the Response Analyzer obtains justifications for beliefs, and also asks what-if queries. These help to fill in gaps involved in a student's seemingly correct response which may have underlying misconceptions. The Deviation-Finder determines correctness of a belief using the following definition:

Definition: A belief $\beta_i = (A \Rightarrow C)$ is correct **iff** $\exists$ a reasoning path ρ_j such that ρ_j starts at A and ends at C. Otherwise (ρ_j does not end at C), β_i is incorrect. All intermediate points in ρ_j are regarded as the student's assumptions and are passed on to the Tutor to query and ascertain β_i's completeness.

The Deviation Finder performs its functions in four major phases. First, it computes the set ($\mathcal{BA}$) of paths starting from a belief's antecedent A. Next it labels the belief by determining if there exist any path(s) in $\mathcal{BA}$ concluding C at some point. The next stage is to identify misconceptions. To do this, it computes the set ($\mathcal{BCONC}$) of paths leading to the belief's conclusion, C, after which it performs a rigorous analysis of $\mathcal{BA}$ and $\mathcal{BCONC}$ to identify embedded misconceptions. The first major function is described in algorithm 6.12 and 6.13, while the others follow in Algorithms 6.14 and 6.15.

Algorithm 6.12: Computation of paths from belief antecedent

Input: β: A belief
$\mathcal{KB}$: Set of Rules (Knowledge-Base)

Process: Forward chain through $\mathcal{KB}$ and compute paths leading from belief's antecedent

Variables:

$Ante_\beta$: β's Antecedent
$Conc_\beta$: β's Conclusion
i, j, m: loop variables
τ: a rule ($\tau \in \mathcal{KB}$)
RC: Rule-specific conclusion
RPREC: Rule-specific preconditions
PPREC: Principal precondition
nextprec: Precondition for chaining next point in a path
$\mathcal{DS}$: Disorder Set (set of all disorders in study domain)
RP: Recurrent Precondition
$\mathcal{SL}$: Search List (subset of $\mathcal{KB}$)
PB: Path Buffer
INC-JUSTS: an array of incomplete justifications or paths
INC-LISTS: an array of search lists for INC-JUSTS
npaths: number of alternative paths
ipcnt: number of incomplete paths
ρ: a path
τ: a rule in $\mathcal{KB}$ or $\mathcal{SL}$
update-BA-paths: path update procedure (see **Algorithm 6.13**)

Output: $\mathcal{BA}$: set of paths leading starting from belief's antecedent, Ante

procedure compute-BA( ) {uses belief antecedent}

```
begin
    npaths = ipcnt := 0                                  {initialize variables}          (1)
    𝒜 = INC-JUSTS = INC-LISTS = ∅
    for each rule τ_i ∈ 𝒦ℬ do                                                            (2)
      begin
        if Ante_β ∈ RPREC_τi then                        {beginning of a major path}     (3)
          begin
            PB := [RPREC_τi, RC_τi]                      {initialize path buffer}        (4)
            nextprec := RC_τi
            RP := Ante_β
            ℒ := 𝒦ℬ - (τ_1 ∪ τ_2 ∪ . . . ∪ τ_i)                                          (5)
            update-BA-paths                              {call update procedure}         (6)
            while ∃ some new ρ_k's (j=1 to m), ⊆ INC-JUSTS do                            (7)
              for each ρ_k do                                                            (8)
                begin
                ℒ := INC-LISTS_ρk
                PB := INC-JUSTS_ρk
                RP := PPREC_τ1
                nextprec := RCτ_1        {initialize next-prec to RC of ℒs}            (9)
                                         { first rule }
                update-BA-paths;                                                        (10)
                 end; {for ρ_k}
            end; {while}
          end; {if}
    end; {for τ_i}
end {compute-BA}
```

The above algorithm is very similar to Algorithm 6.5, with the major exception that this time we are searching in a forward direction, using rule preconditions, instead of rule conclusions, as links to succeeding path points. Similarly, **update-BA-paths** is very similar to Algorithm 6.6, but different for the same reasons. Essentially, the algorithm produces a set of paths ($\mathcal{BA}$) emanating from the student's belief's antecedent ($Ante_\beta$) (1). Looping through the Knowledge-Base, it looks for all rules in which $Ante_\beta$ holds in the precondition set (2)-(3). If such a rule τ_i is found, its preconditions and conclusions are used to initialize the path buffer, PB, and its conclusion is used as the

chaining precondition to the next path point (4). RP refers to 'recurrent precondition', and is used to keep track of multiple occurrences of a precondition in more than one rule. Such multiple occurrences are used as starting points for future chains. A new search list, $\mathcal{SL}$, is then set to the rest of the Knowledge-Base from τ_{i+1} (5). These variables (PB, nextprec, RP, $\mathcal{SL}$) are then transmitted to the **update-BA-paths** procedure for further chaining (6). On return from **update-BA-paths**, incomplete paths from INC-LISTS are completed by setting the variables to the appropriate starting point for each incomplete path retrieved from INC-LISTS and INC-JUSTS (7)-(10). INC-LISTS contains $\mathcal{SL}$'s to be searched for each incomplete path, while INC-JUSTS contains the corresponding path-buffers or justifications up to the point at which the incomplete path was stored. Incomplete paths are necessarily created when a precondition occurs in more than one rule. Paths up to new occurrences are stored as incomplete paths, which are completely chained after the chain from the first occurrence has been completed.

Algorithm 6.13: Update-BA-paths procedure

Input: nextprec: Precondition used to chain the next point in a path
PB: Path Buffer
RP: Recurrent Precondition
$\mathcal{SL}$: Search List

Process: Connect new points to path
Create new subsidiary paths

Variables:
RC: rule-specific conclusion
RPREC: rule-specific preconditions
PPREC: Principal precondition in rule
Conc_β: Belief's conclusion
k: loop variable
INC-JUSTS: an array of incomplete justifications or paths
INC-LISTS: an array of search lists for INC-JUSTS
npaths: number of alternative paths
ipcnt: number of incomplete paths
ρ: a path in INC-PATHS
τ: a rule in $\mathcal{KB}$ or $\mathcal{SL}$

$\mathcal{DS}$: Disorder-Set (of disorders in domain)
Output: Updated INC-PATHS, INC-LISTS, $\mathcal{BA}$

```
procedure    update-BA-paths( )
  begin
    for each rule τk ∈ SL do                                          (1)
      begin
        if nextprec ∈ RPREC_τk then                                   (2)
          begin                                  {Connect new point}
            PB := PB ∪ [RPREC_τj , RC_τk]
            if (RC_τk ∈ DS) or (RC_τk = Conc_β)                       (3)
                                   {path reaches a final disorder}
              then begin           {end of path}                      (4)
                     npaths := npaths + 1
                     BA_npaths := PB
                     PB := PB - [RPREC_τk , RC_τk]
                                   {make room for more end points}
                   end;
              else nextprec := RC_τk;                                 (5)
          end; {if nextprec}
        if PPREC_τk = RP then          {Create subsidiary path}       (6)
          begin
            ipcnt := ipcnt + 1
            INC-JUSTS_ipcnt := PB ∪ [RPREC_τk , RC_τk]                (7)
            INC-LISTS_ipcnt := SL - (τ1 ∪ τ2 ∪ . . . ∪ τk-1)          (8)
          end; {if}
      end; {for τk}
  end {update-BA-paths}
```

This is very similar to Algorithm 6.6, except again that the search this time is data-driven, using a precondition rather than a conclusion, to chain rules in paths. The conditions for reaching the end of a path here are: the current rule concludes a proper disorder rather than an intermediate conclusion, or the current rule concludes the student's own belief-conclusion $Conc_\beta$ (3).

Algorithm 6.14: Belief labeling

Input: $\mathcal{BA}$: Set of paths leading from $Ante_\beta$

Process: Analyze paths in $\mathcal{BA}$ and label current belief

Variables:

β: A single belief $\{\beta \in \mathcal{B}\}$

ρ: a path in $\mathcal{BA}$

$Tail_\rho$: The last item stored in path p

$Ante_\beta$: Belief's Antecedent

$Conc_\beta$: Belief's conclusion

i: loop variable

cnt: counter

$\mathcal{BJUST}$: Belief Justification (set of correct paths from A)

Label: Belief's label

Output: Labeled beliefs:

- $\mathcal{CB}$: Correct Beliefs {All beliefs β_i | $Label_{\beta_i}$ = 'Correct'}
- $\mathcal{ICB}$: Incorrect Beliefs {All beliefs β_k | $Label_{\beta_k}$ = 'Incorrect'}

$\mathcal{BA}$: Set of chains of beliefs, each starting from a $\mathcal{B}$elief's $\mathcal{A}$ntecedent

```
procedure    label-beliefs( )
  begin
    for each path ρi ∈ BA do                                   (1)
      if Tail_ρi = Conc_β then                                 (2)
        begin
          cnt := cnt + 1                                       (3)
          BJUST_cnt := ρi                                      (4)
        end; {if / for}
    if BJUST ≠ ∅ then                                          (5)
      begin
        Label_β := 'Correct'                                   (6)
        CB := CB ∪ β                                           (7)
        BA := BJUST                                            (8)
      end
      else begin
        Label_β := 'Incorrect'                                 (9)
        ICB := ICB ∪ β                                         (10)
      end; {if}
  end; {label-beliefs}
```

Algorithm 6.14 is employed by the Deviation Finder to label a belief as correct or incorrect, by determining whether there exist any paths in the belief's $\mathcal{BA}$ computed in Algorithms 6.12 and 6.13. Each path in $\mathcal{BA}$ is analyzed by the algorithm (1). $Tail_{\rho_i}$

refers to the last conclusion in the current path ρ_i (2). If the path ρ_i ends in the student's belief-conclusion (that is, $\text{Tail}_{\rho_i} = \text{Conc}_\beta$), then ρ_i is added to the set of justifications (*JUST*) for the current belief (3)-(4). *JUST* serves to inform the Tutor that the student's current belief is labeled correct for the reason that those paths do exist from the Knowledge-Base. The path analysis is done for all ρ_i. At the end of the path analysis, *JUST* either contains some justifying paths for a correct belief, or is empty for an incorrect belief. If *JUST* is not empty, then the belief is labeled correct, added to the set *CB* of correct beliefs to be sent to the **instructional-belief-update** algorithm (6.16), and *BA* is reset to contain only the correct paths in *JUST* (6)-(8). If *JUST* contains no justifying paths, the belief is labeled incorrect and is added to the set of incorrect beliefs, *ICB* (9)-(10).

Example 6.4

Suppose the student believes incorrectly that

(equal stress-response 'psychotic) $\Rightarrow$ delusional-disorder

Using the graph in Figure 6.2, the set *BA* of paths emanating from the antecedent is the following:

ρ_1: (Rule-10 Rule-14)
ρ_2: (Rule-10 Rule-15)

In detail, these are:

ρ_1: ((equal non-organic-disorder T) ^ (<duration 4) ^ (equal schizophrenic-prodromal-symptoms nil) ^ (equal emotional-turmoil T) ^ (equal stress-response 'psychotic)), light-moods) *{ for Rule-10}*
((equal light-moods T) ^ (equal full-mood-syndrome T)) psychotic-mood-disorder)) *{for Rule-14}*

ρ_2: ((equal non-organic-disorder T) ^ (<duration 4) ^ (equal schizophrenic-prodromal-symptoms nil) ^ (equal emotional-turmoil T) ^ (equal stress-response 'psychotic)), light-moods) *{ for Rule-10}*
((equal light-moods T) ^ (equal full-mood-syndrome nil)) brief-reactive-psychosis)) *{for Rule-15}*

Checking the Tail of each path, the Deviation Finder discovers that none ends in the student's conclusion, {delusional-disorder}. $\mathcal{BJUST}$ is therefore empty, and the belief is labeled as incorrect.

Algorithm 6.15: Deviation Finder's trace of misconceptions

The algorithm described here is performed on incorrect beliefs. First, chains are traced involving belief conclusions ($\mathcal{BCONC}$). This is a backward chaining procedure very similar to the Knowledge Finder's algorithms 6.5 and 6.6. Therefore, to avoid repetition of algorithms, we will obtain $\mathcal{BCONC}$ by making a procedure call to the Knowledge Finder's **compute-alternative-strategies**. $\mathcal{BCONC}$ is compared to chains obtained in Algorithms 6.12 and 6.13 above ($\mathcal{BA}$). This is a rigorous comparison, whose results may fall between two extremes.

Worst case: Chains are mutually exclusive or disjoint, that is, no two paths ρ_i in $\mathcal{BA}$, ρ_j in $\mathcal{BCONC}$, are similar.

Best case: Chains differ minimally, that is, most beliefs are similar except for few easily identifiable differences.

Secondly, $\mathcal{BCONC}$ and $\mathcal{BA}$ are intersected to identify common beliefs ($\mathcal{BAC}$) which are removed from each set. It is necessary to remove intersecting sub-beliefs, since we are looking for misconceptions, and intersecting sub-beliefs imply points of agreement, rather than points of disparity, between $\mathcal{BA}$ and $\mathcal{BCONC}$. Next, keyword checks are done on remaining beliefs in both sets (that is, {$\mathcal{BA}$ - $\mathcal{BAC}$} and {$\mathcal{BCONC}$ - $\mathcal{BAC}$}), to find common keywords with different values in each set. These are marked as sources of possible misconceptions. Trace of misconceptions is done for incorrect beliefs only, that is, beliefs β with $\mathcal{BA}$ sets such that $\forall\ \rho_i \in \mathcal{BA}$, ρ_i ends at some point other than the belief's conclusion, $Conc_\beta$.

Input: $\mathcal{BA}$: Set of paths leading from $Ante_\beta$
$\mathcal{KB}$: Knowledge-base

Process: Pass through $\mathcal{KB}$ and identify misconceptions

Variables: SB: Strategy-base obtained from Algorithm 6.5.
ρ_{ba}: Path in $\mathcal{BA}$
ρ_{bconc}: Path in $\mathcal{BCONC}$
pbac: Intersection of ρ_{ba} and ρ_{bconc}
prba: Remainder of ρ_{ba} after removing pbac
prbc: Remainder of ρ_{bconc} after removing pbac
suba: A sub-belief in prba
subc: A sub-belief in prbc
prec1: The precondition part of suba
prec2: The precondition part of subc
keyword: The keyword involved in a precondition
cnt: A counter

Output: $\mathcal{BCONC}$: Set of chains of beliefs, each ending at a $\mathcal{B}$elief's $\mathcal{CONC}$lusion
$\mathcal{MISC}$: Set of misconceptions

```
procedure   Identify-misconceptions( )
  begin
    compute-alternative-strategies(Conc_β)   {Trace chains involving}   (1)
                                             { belief conclusion}
    BCONC := SB                                                         (2)
    for each path ρ_ba ∈ BA do                                          (3)
      begin
        for each path ρ_bconc ∈ BCONC do                                (4)
          begin
            pbac := ρ_ba ∩ ρ_bconc          {intersecting sub-beliefs}  (5)
            prba := ρ_ba - pbac             {rest of ρ_ba}              (6)
            prbc := ρ_bconc - pbac          {rest of ρ_bconc}           (7)
            for each sub-belief suba ∈ prba do                          (8)
              for each prec1 ∈ Prec-part_suba do                        (9)
                for each sub-belief subc ∈ prbc do                      (10)
                  for each prec2 ∈ Prec-part_subc do                    (11)
                    if (keyword_prec1 = keyword_prec2) and              (12)
                      (value_keyword_prec1 ≠ value_keyword_prec2) then
                      begin
                        cnt := cnt + 1
                        MISC_cnt := (prec1, prec2)
```

```
                    print (prec1, prec2)
                                    {as possible source of misconception}
                end; {if}
            end; {for prec2}
        end; {for subc}
    end; {for prec1}
end; {for suba}
end; {for ρbconc}
end; {for ρba}
end {Identify-misconceptions}
```

This algorithm first calls **compute-alternative-strategies** (Algorithm 6.5), which computes the set of paths ending at the current belief's conclusion, $Conc_\beta$ (1). The result is assigned to $\mathcal{BCONC}$ (2). Then looping through the paths in $\mathcal{BA}$ (3), it compares each path ρ_{ba} in $\mathcal{BA}$ with each path ρ_{bconc} in $\mathcal{BCONC}$ (4). The aim is to obtain an intersection of the two paths, pbac (5), leaving prba in $\mathcal{BA}$ and prbc in $\mathcal{BCONC}$ (6)-(7). Intersections would be sub-beliefs where the preconditions and conclusions are identical in the two paths under consideration. Next, for each remaining sub-belief suba in prba, the algorithm compares each precondition (prec1) with each precondition (prec2) of each sub-belief (subc) of prbc (8)-(11). The purpose of this is to find symptom or variable names which are common to both paths, but differ in values. The difference in values would account for why the student believes one thing while the Knowledge-Base represents another. The algorithm therefore looks at the precondition part of each sub-belief (recall that each path point is a pair [X, Y] meaning 'X $\Rightarrow$ Y'), and identifies common keywords (12). Where a common keyword is found whose values differ, the values of the two occurrences are recorded as possible sources of a misconception. For instance, if we obtain (Manic-syndrome = T) and (Manic-syndrome = Nil) in suba and subc, respectively, then we can reasonably infer that the student may be having a problem with understanding the relationship of 'Manic-syndrome' with the problem at hand. The set of misconceptions, $\mathcal{MISC}$ is transmitted to the Tutor for further query, to compute updated student beliefs. The overall loop (3)-

(12) is done for the entire sets of paths in *BA* and *BCONC*, resulting in an m:n analysis for all m and n paths in *BA* and *BCONC*, respectively, as shown in Figure 6.9.

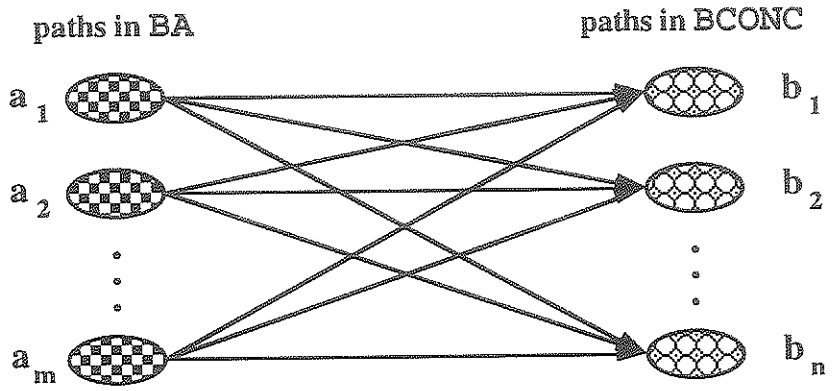


Figure 6.9. Interaction between paths in *BA* and *BCONC*

Example 6.5

The student-belief in Example 6.4 ((equal stress-response 'psychotic) ⇒ delusional-disorder) qualifies for this algorithm, since it is labeled 'Incorrect'. The Deviation Finder computes BCONC, which is the set of paths ending at {delusional-disorder}. Two rules conclude delusional disorder: Rule-36 and Rule-37. The paths ending at these rules are the following:

(Rule-37 Rule-32 Rule-17 Rule-13 Rule-09 Rule-07)
(Rule-37 Rule-32 Rule-17 Rule-13 Rule-09 Rule-06)
...
(Rule-37 Rule-32 Rule-17 Rule-13 Rule-09 Rule-01)
(Rule-37 Rule-32 Rule-17 Rule-12 Rule-09 Rule-07)
(Rule-37 Rule-32 Rule-17 Rule-12 Rule-09 Rule-06)
...
(Rule-37 Rule-32 Rule-17 Rule-12 Rule-09 Rule-01)
(Rule-37 Rule-32 Rule-17 Rule-11 Rule-09 Rule-07)
(Rule-37 Rule-32 Rule-17 Rule-11 Rule-09 Rule-06)
...
(Rule-37 Rule-32 Rule-17 Rule-11 Rule-09 Rule-01)
(Rule-37 Rule-31 Rule-17 Rule-13 Rule-09 Rule-07)
(Rule-37 Rule-31 Rule-17 Rule-13 Rule-09 Rule-06)
...
(Rule-37 Rule-31 Rule-17 Rule-13 Rule-09 Rule-01)

(Rule-37 Rule-31 Rule-17 Rule-12 Rule-09 Rule-07)
(Rule-37 Rule-31 Rule-17 Rule-12 Rule-09 Rule-06)
° ° °
(Rule-37 Rule-31 Rule-17 Rule-12 Rule-09 Rule-01)
(Rule-37 Rule-31 Rule-17 Rule-11 Rule-09 Rule-07)
(Rule-37 Rule-31 Rule-17 Rule-11 Rule-09 Rule-06)
° ° °
(Rule-37 Rule-31 Rule-17 Rule-11 Rule-09 Rule-01)
(Rule-36 Rule-32 Rule-17 Rule-13 Rule-09 Rule-07)
(Rule-36 Rule-32 Rule-17 Rule-13 Rule-09 Rule-06)
° ° °
(Rule-36 Rule-32 Rule-17 Rule-13 Rule-09 Rule-01)
(Rule-36 Rule-32 Rule-17 Rule-12 Rule-09 Rule-07)
(Rule-36 Rule-32 Rule-17 Rule-12 Rule-09 Rule-06)
° ° °
(Rule-36 Rule-32 Rule-17 Rule-12 Rule-09 Rule-01)
(Rule-36 Rule-32 Rule-17 Rule-11 Rule-09 Rule-07)
(Rule-36 Rule-32 Rule-17 Rule-11 Rule-09 Rule-06)
° ° °
(Rule-36 Rule-32 Rule-17 Rule-11 Rule-09 Rule-01)
(Rule-36 Rule-31 Rule-17 Rule-13 Rule-09 Rule-07)
(Rule-36 Rule-31 Rule-17 Rule-13 Rule-09 Rule-06)
° ° °
(Rule-36 Rule-31 Rule-17 Rule-13 Rule-09 Rule-01)
(Rule-36 Rule-31 Rule-17 Rule-12 Rule-09 Rule-07)
(Rule-36 Rule-31 Rule-17 Rule-12 Rule-09 Rule-06)
° ° °
(Rule-36 Rule-31 Rule-17 Rule-12 Rule-09 Rule-01)
(Rule-36 Rule-31 Rule-17 Rule-11 Rule-09 Rule-07)
(Rule-36 Rule-31 Rule-17 Rule-11 Rule-09 Rule-06)
° ° °
(Rule-36 Rule-31 Rule-17 Rule-11 Rule-09 Rule-01),

a total of eighty-four (84) paths. The next stage is for the Deviation Finder to compare each path in *BCONC* with the two *BA* paths obtained in Example 6.4. Let us expand one of the above paths, say

ρ_{bconc} = p1 = (Rule-37 Rule-32 Rule-17 Rule-13 Rule-09 Rule-07)

= ((equal psycho-delusions-3 T) ^ (equal behavior (not 'bizarre)) ^ (equal visual-hallucinations T)), delusional-disorder) *{Rule-37}*

((equal psycho-delusions-1 T) ^ (< mood-duration psychotic-disturbance-duration)), psycho-delusions-3) *{Rule-32}*
((equal strong-moods T) ^ (equal schizophrenic-prodromal-symptoms nil)), psycho-delusions-1) *{Rule-17}*
((equal non-organic-disorder T) ^ (>= duration 4) ^ (equal stress-response (not 'psychotic))), strong-moods) *{Rule-13}*

((equal psychotic-symptoms-positive T) ^ (equal organic-factor nil)), non-organic-disorder) *{Rule-09}*
((equal behavior 'grossly-disorganized), psychotic-symptoms-positive) *{Rule-07}*

We shall compare this with the first path in $\mathcal{BA}$,

ρ_{ba} = ((equal non-organic-disorder T) ^ (< duration 4) ^ (equal schizophrenic-prodromal-symptoms nil) ^ (equal emotional-turmoil T) ^ (equal stress-response 'psychotic)), light-moods) *{ for Rule-10}*
((equal light-moods T) ^ (equal full-mood-syndrome T)) psychotic-mood-disorder)) *{for Rule-14}*

An intersection of ρ_{ba} and ρ_{bconc} yields $\rho_{bac} = \emptyset$.
Next, we do a keyword search with the following results:

Table 6.1. Results of keyword search for ρ_{ba} and ρ_{bconc}

Keyword	Value in ρ_{ba}	Value in ρ_{bconc}	Possible source of misconception?
non-organic-disorder	T	T	No
duration	< 4	>= 4	Yes
schizophrenic-prodromal-symptoms	nil	nil	No
emotional-turmoil	T	-	No
stress-response	'psychotic	not 'psychotic	Yes
light-moods	T	-	No
full-mood-syndrome	T	-	No

The resulting $\mathcal{MISC}$ includes 'duration' and 'stress-response' as possible sources of the student's misconceptions. The algorithm therefore yields a correct result since stress-response would yield different values if the student understood the relationship between its psychotic or non-psychotic values to the problem at hand. Note that the algorithm sometimes produces results which may not be misconception-sources for the student. It is left to the Tutor to query and sift out non-problem areas.

Notice also that the algorithm stops after searching with all the preconditions in ρ_{ba}, even though ρ_{bconc} still has other 'untouched' preconditions. It is not necessary to compare preconditions in ρ_{bconc} which are not in ρ_{ba}, since they would yield a 'No' in Column 4 of Table 6.1. above. We could reduce the search by first counting the numbers of preconditions in each path, and conducting the search with the smaller path. On the other hand, we could avoid this approach since the computational cost of counting preconditions may level out with the cost of searching for excess preconditions.

6.9.1 Missing beliefs

Identification of missing beliefs can be done by the Deviation Finder by analyzing the solution obtained by the Expert Problem Solver and locating which beliefs or sub-paths are not included in the student's current beliefs about the problem (which are collected in the Response Analysis phase). This identification can be achieved by an iterative loop involving a single test as follows:

if $\beta_i \notin DD_{EPS}$ **then** $\mathcal{MISS} := \mathcal{MISS} \cup \beta_i$

for each belief β_i, where $\mathcal{MISS}$ is the list of missing beliefs which will be passed on to the Tutor after the modeling process, to be addressed in further interactions with the student.

6.10 Belief-revision: The Instructional Belief-Update Approach

The Deviation-Finder's labeled beliefs computed above must be properly combined with existent beliefs to maintain an accurate representation of the student's overall beliefs, without inconsistencies. The maintenance of such a belief system is performed by the Belief Revision Unit, using the approach described in Section 6.4. There are two belief-bases, necessitated by the fact that we label the student's beliefs as correct or incorrect. Before any new belief is stored in its destination belief base, the Belief-Revision Unit

looks for, and removes, contradictions whose presence would render the belief-bases inaccurate. Examples of contradictions are discussed under Algorithm 6.17. Dependent beliefs are also recursively derived for incoming beliefs

Note that it is possible to maintain only one belief-base for both correct and incorrect beliefs, which belief-base would then include each belief's appropriate label (for example a binary field with '1' and '0' for correct and incorrect beliefs, respectively). However, the search procedure with a single belief-base would be about twice as long as when the two classes of beliefs are separated, since all beliefs would be visited to check for their correctness labels. With separation of correct and incorrect beliefs, looking for contradictions would only involve the opposite belief-base for any belief, that is, the algorithms would search only the Correct Belief Base for contradiction of incorrect beliefs, and vice-versa. As explained in Section 6.4.7, we have chosen the use of two belief-bases to reduce computational cost.

Algorithms 6.16 to 6.21 describe the belief update functions performed by the Belief-Revision Unit..

Algorithm 6. 16: Belief Revision

Input: $\mathcal{CB}$: Correct Beliefs {All beliefs β_i | Label_{β_i} = 'Correct'}
$\mathcal{ICB}$: Incorrect Beliefs {All beliefs β_k | Label_{β_k} = 'Incorrect'}
$\mathcal{CBB}$: Correct Belief Base
$\mathcal{ICBB}$: Incorrect Belief Base
Correct-Usage-Counts: number of times each belief is used correctly
Incorrect-usage-counts: number of times each belief is used incorrectly

Process: Remove contradictions in $\mathcal{ICBB}$, to store correct beliefs
Remove contradictions in $\mathcal{CBB}$, to store incorrect beliefs

Variables:

β:	a belief
contra-flag:	flag indicating presence of a contradiction among beliefs
i, j, k, m:	loop variables
find-contradiction:	procedure which detects contradictions

Id: belief's unique identifier
Type: belief's type
Justifier: belief's justifier set
CBB-count: count of beliefs in CBB
ICBB-count: count of beliefs in ICBB
query-derived: procedure that queries for persistence of derived beliefs
generate-derived: procedure that generates new beliefs from an existent one

Output: Updated $\mathcal{CBB}$
Updated $\mathcal{ICBB}$
Updated Correct-Usage-Counts, Incorrect-usage-counts

```
procedure   instructional-belief-update(β)
  begin
    for each βi ∈ CB do                                                    (1)
      begin
       for each βj ∈ ICBB do                  {look for contradictions}    (2)
         begin
           find-contradiction (βi, βj)                                     (3)
           if contra-flag = 1 then            {that is, βj = ¬βi }         (4)
              begin
                 ICBB := ICBB - βj                                         (5)
                 query-derived(βj, ICBB)                                   (6)
              end; {if contra-flag}
         end; {for βj}

         Id_βi := CBB-count + 1               {insert new belief}          (7)
         Type_βi := "OBSERVED"                                             (8)
         Justifier_βi := ∅                                                 (9)
         CBB := CBB ∪ βi                                                   (10)
         generate-derived(βi)                                              (11)
         Correct-Usage-Count_βi := Correct-Usage-Count_βi + 1               (12)
    end; {for βi}
    for each βk ∈ ICB do                                                   (13)
      begin
        for each βm ∈ CBB do              {find contradictions}            (14)
          begin
            find-contradiction(βk, βm)                                     (15)
            if contra-flag = 1 , then     {that is, βm = ¬βk }             (16)
               begin
```

$\mathcal{CBB} := \mathcal{CBB} - \beta_m$ (17)

query-derived(β_m, $\mathcal{CBB}$) (18)

end; {if contra-flag}

end; {for β_m}

Id_{β_k} := CBB-count + 1 {insert new belief} (19)

$Type_{\beta_k}$:= "OBSERVED" (20)

$Justifier_{\beta_k} := \varnothing$ (21)

$\mathcal{ICBB} := \mathcal{ICBB} \cup \beta_k$ (22)

generate-derived(β_i) (23)

Correct-Usage-$Count_{\beta_k}$:= Correct-Usage-$Count_{\beta_k}$ - 1 (24)

Incorrect-usage-$count_{\beta_k}$:= Incorrect-usage-$count_{\beta_k}$ + 1 (25)

end; {for β_k}

end {instructional-belief-update}

Algorithm 6.16 performs the belief revision process surrounding the entire Student Modeler. The Instructional Belief Update Approach works by examining the Incorrect Belief Base ($\mathcal{ICBB}$) before storing a correct belief, and examining the Correct Belief Base ($\mathcal{CBB}$) before storing an incorrect belief. First, for each correct belief β_i (1) labeled by the Deviation Finder in Algorithm 6.14, the Belief Revision Unit locates any β_j which contradicts β_i (3). If a contradiction is found, β_j is deleted from $\mathcal{ICBB}$, and then β_i is added to the $\mathcal{CBB}$ (4) - (11). For each deleted belief, the system must query the student to determine whether beliefs derived from the deleted belief may still be retained in the affected belief base. This task is performed by the **query-derived** function (6) in Algorithm 6.20. Also, for each belief added to a belief base, the system must examine the INFERENCE-RULEBASE to derive new beliefs which may be derivable from the new belief (11). This is performed by the **generate-derived** function described in Algorithm 6.21. The Correct-Usage-Count for β_i is then incremented (12). A similar procedure is performed for the addition of incorrect beliefs to the $\mathcal{ICBB}$, by first deleting any contradictory beliefs in the $\mathcal{CBB}$ (13)-(25). This

time though, the Incorrect-usage-count is incremented for the current belief. *Correct-Usage-Count* measures the number of times each belief has been applied correctly by the student. It is similar to the USE-HISTORY of GUIDON (Clancey, 1987), and serves to advise the Modeler as to whether the student is having a big problem with a particular belief, or whether the student has a simple misunderstanding (or made a simple slip) with respect to the query at hand. For instance, if a student has correctly applied a belief, say 100 times, and then misapplies it the next time a query is posed, the Modeler may suppose that the student made a simple mistake, or may assume some misunderstanding which may be explained briefly. The *Incorrect-usage-count* variable keeps count of incorrect applications. A high *Incorrect-usage-count* suggests that the student does have a problem with the specific belief. The aim of the Tutor is to minimize the percent growth rate of *Incorrect-usage-counts*, and maximize the percent growth rate of *Correct-Usage-Counts*, where a reasonable growth function could be

$$f(x) = \frac{100x}{\text{total number of applications}} .$$

The challenge in the above algorithm is the location of contradictions (3) a procedure performed by **find-contradiction** in Algorithm 6.17.

Algorithm 6.17. Location of belief contradictions

Input: Two beliefs, β_1 and β_2

Process: Look for contradictions in precondition parts of beliefs
If no contradiction in preconditions, look for contradictions in conclusion parts

Variables:

prec:	precondition part of a belief
conc:	conclusion part of a belief
b:	a belief
return-value:	value returned from a contradiction check
prec-return:	return value for preconditions
conc-return:	return value for conclusions
check-parts:	procedure doing sub-belief checks

Output: contra-flag: flag indicating whether b1 contradicts b2

```
procedure    find-contradiction(β1, β2)
  begin
    contra-flag = 0                    {initialize contradiction flag}          (1)
    check-parts(prec_β1, prec_β2)      {look for contradictions in preconditions} (2)
    prec-return = return-value                                                  (3)
    if prec-return = 1, then                                                    (4)
          begin
            contra-flag = 0
            return;
          end
    check-parts(conc_β1, conc_β2) {look for contradictions in conclusions}      (5)
    if return-value = 1, then contra-flag = 1                                   (6)
  end; {find-contradiction}
```

Algorithm 6.17 detects contradictions among two beliefs, β_1 and β_2. The idea applied here was explained using a truth table in Section 6.4.5 and repeated here for emphasis.

A	B	$A \Rightarrow B$
T	T	T
T	F	F
F	T	T
F	F	T

Therefore given a belief

$\beta_i = (A \Rightarrow Z)$,

another belief β_k is seen as a contradiction of β_i (that is, $\neg\beta_i$) if and only if β_k is of the form

- $A \Rightarrow \neg Z$

Therefore given two beliefs β_1 and β_2, as follows:

$\beta_1 = prec_{\beta_1} \Rightarrow conc_{\beta_1}$

$\beta_2 = prec_{\beta_2} \Rightarrow conc_{\beta_2}$,

this algorithm aims at detecting whether

$prec_{\beta_2} = prec_{\beta_1}$, **and**

$conc_{\beta_2} = \neg conc_{\beta_1}$.

If this conditions holds, then it is concluded that β_1 and β_2 are contradictions. As was shown in the implication truth table, if $prec_{\beta_2} = \neg prec_{\beta_1}$, then the two beliefs are automatically not contradictions and it would not be necessary to check for the relationship between the conclusion parts. The first check is therefore aimed at determining the relationship between the two preconditions. If they are identical, then conclusions are checked for conflict. A contradiction of the overall belief is confirmed if the two conclusions are not the same.

In the algorithm, contra-flag is first set to zero (1), indicating that it is initially assumed that no contradiction exists. A procedure named **check-parts** (Algorithm 6.18) is then called to determine if the precondition parts of the two beliefs contradict each other (2). **Check-parts** returns 'return-value', which becomes prec-return, to indicate contradiction status of the preconditions of the two beliefs (3). If the preconditions contradict each other (that is, by yielding a **False** in the comparison performed by **check-parts**), then the procedure automatically returns with a 'non-contradiction' indicator. Otherwise, it proceeds to check the conclusion parts by calling **check-parts** in (5). The resulting contradiction status of β_1 and β_2 is determined (6). The only condition under which a contradiction flag is set is if the conclusion parts contradict each other when the preconditions do not, as shown in (6).

Algorithm 6.18. Looking for contradictions in sub-beliefs

Input: Two belief parts, X and Y

Process: Look for contradictions in X and Y

Variables:

data-type: the system data-type of a variable-name
X: first item in check
Y: second item in check
check-non-numeric: another procedure doing sub-belief checks on non-numeric variables

keyword: keyword involved in X or Y

Output: return-value: value indicating presence or absence of contradiction

```
procedure    check-parts(X, Y)
  begin
    return-value = 0                    {initialize to 'non-contradiction'}      (1)
    if  data-type(keyword_X) = ¬'numeric'  or  data-type(keyword_Y) = ¬'numeric'
                                                                                 (2)
        then
          begin
            check-non-numeric                                                    (3)
            return(return-value)
          end;
        else begin
          if keyword_X ≠ keyword_Y                                               (4)
            then return(return-value)                                            (5)
            else begin
              eval X                                                             (6)
              if (eval Y) = F                                                    (7)
                  then   return-value = 1
              return(return-value)                                               (8)
          end; {else}
        end; {if data-type}
  end; {check-parts}
```

Check-parts determines whether two sub-beliefs X and Y (which may be both preconditions or conclusions) are contradictory. It returns a single value, *return-value*, to **find-contradiction**. First, return-value is zero, assuming a non-contradiction (1). Then the algorithm checks the data-type of the keywords involved in X and Y (2). If they are non-numeric, **check-non-numeric** is called (3). We have assumed the data-type of any variable is implicitly declared in the implementation environment. Separation of the procedures for numeric and non-numeric variables was done because the evaluation approaches are different. For non-numeric sub-beliefs (for example, (equal social-effect 'impaired)), all we need do is look at the value of the keyword. For numeric sub-beliefs most of which may come in ranges (for example (>= continuous-par-time 15)), we have to perform a numeric evaluation of the entire sub-belief. If the

keywords are both numeric, the algorithm determines whether they are the same keyword (4). If they are not, the procedure is terminated. Otherwise, the two sub-beliefs are evaluated. If the second is false with respect to the first, then a contradiction has been identified (6) - (8).

Algorithm 6.19. Looking for contradictions in non-numeric sub-beliefs

Input: Two belief parts, X and Y

Process: Look for contradictions in X and Y

Variables:
X: first item in check
Y: second item in check
keyword: keyword involved in X or Y
value-part: the value part of sub-belief
relationship: matrix indicating inter-variable relationships

Output: return-value: value indicating presence or absence of contradiction

```
procedure check-non-numeric(X, Y)
  begin
    if keyword_X = keyword_Y                    {if same keyword but}          (1)
     then if value-part_X ≠ value-part_Y        {different values, then it is} (2)
            then return-value = 1;              {a contradiction}              (3)
     else if (relationship(keyword_X, keyword_Y) = 'Not')                      (4)
            and (value-part_X = value-part_Y)
                  then return-value = 1                                        (5)
            {if different keywords, but keywords are opposites and}
            {have same value, then it is a contradiction}
  end; {check-non-numeric}
```

Algorithm 6.19 performs contradiction check on non-numeric sub-beliefs, X and Y. If the keywords of both sub-beliefs are the same (1), then the values are checked. If the values are not the same, then a contradiction is implied (2)-(3). If the keywords are not the same, then we check the Variable-Relationship matrix, to determine if the two variables have a 'Not' relationship (4). The Variable-Relationship matrix in this design is a two-dimensional square matrix containing all variable-names in the design. A 'Not' in $VR_{i,k}$ indicates that $variable_i = \neg variable_k$. For examples, *light-moods* is regarded

as an opposing variable to *strong-moods*. Therefore $VR_{\text{light-moods, strong-moods}}$ = 'Not'. If therefore these two variable names occur separately in X and Y of Algorithm 6.19, and have the same value-parts, a contradiction is implied.

Algorithm 6.20. Querying student for persistence of derived beliefs

Input: β: a belief
BB: a belief base (representing $\mathcal{CBB}$ or $\mathcal{ICBB}$)

Process: Look for all beliefs gamma derived from β, and query on them
Retain each γ in BB if student still holds gamma
For each retained γ, change status to "OBSERVED"

Variables: γ: a belief
Type: belief type
Justifier: belief's justifier set
response: response obtained as to belief's persistence with student

Output: modified BB

```
procedure    query-derived(β, BB)
  begin
    for each γ ∈ BB do                                              (1)
      if (Type_γ = "DERIVED") and (β ∈ Justifier_γ) then             (2)
        begin
          query-to-confirm(γ, response)                             (3)
          if response = 1 then
            begin
              Type_γ = "OBSERVED"                                   (4)
              Justifier_γ = ∅                                       (5)
              generate-derived(γ, BB)                               (6)
            end;
            else begin
                BB := BB - γ                                        (7)
                query-derived(γ, BB)                                (8)
            end;  {if response}
        end;  {if}
    end;  {for γ}
  end  {query-derived}
```

Algorithm 6.20 is a recursive function which queries the student to determine if the student still holds a belief γ derived from a previously deleted belief β. In (1), the algorithm examines the belief base BB in which β was located. It looks for all derived beliefs which include β in their justifier sets (2). It then queries the student to confirm if the derived belief persists in his/her beliefs. This is done using **query-to-confirm** (3), a query procedure which we assume asks a simple question and obtains a response in a "Yes or No" form from the student. A "Yes" would signify a "1" for the *response* variable, and a "No" would signify "0". If the student still holds belief γ, then its status and justifier set are modified in (4) and (5). Otherwise, it is deleted from the belief base (7). For a belief whose status is changed to "OBSERVED", the **generate-derived** procedure must be called (6) to generate its own dependent beliefs. The **query-derived** procedure is recursive in itself since it is called for every derived belief which is not persistent, or which is deleted after the query process (8).

Algorithm 6.21. Generating new beliefs from existent beliefs

Input: β: an observed belief
BB: a belief base (representing $\mathcal{CBB}$ or $\mathcal{ICBB}$)

Process: Look for rules in the INFERENCE-RULEBASE whose antecedent is the conclusion part of β
Use such rules to generate new beliefs

Variables: γ: a derived belief
Id: belief's identifier
Type: belief's type
Justifier: belief's justifier set
belief-count: count of beliefs in BB
CBB-count: count of beliefs in $\mathcal{CBB}$
ICBB-count: count of beliefs in $\mathcal{ICBB}$

```
procedure    generate-derived(β, BB)
  begin
    for each inf ∈ INFERENCE-RULEBASE do                          (1)
      if Prec-part_inf = Conc_β then                              (2)
        begin
          if BB = CBB then                                        (3)
```

```
                begin
                    belief-count := CBB-count                    (4)
                    target-base := CBB
                end;
                else begin
                        belief-count := ICBB-count               (5)
                        target-base := ICBB
                end; {if}

            Id_γ := belief-count                                 (6)
            Type_γ := "DERIVED"                                  (7)
            Antecedent_γ := Prec_β                               (8)
            Conclusion_γ := Conc-part_inf                        (9)
            Justifier_γ := Justifier_γ ∪ {Id_β, Id_inf}          (10)
            target-base := target-base ∪ γ                       (11)
            instructional-belief-update(γ)                       (12)

        end; {if}
    end; {for inf}
end {generate-derived}
```

Algorithm 6.21 recursively generates new beliefs from older beliefs. It looks for rules in an INFERENCE-RULEBASE (1) whose precondition parts are identical to the conclusion part of the current belief β (2). Thus given a belief (β: A ⇒ C) and an inference rule (inf: C ⇒ D), **generate-derived** generates a new belief of the form (γ: A ⇒ D). The algorithm ensures that it is adding to the correct belief base by checking in (3) and consequently setting the current count of beliefs appropriately (4)-(5). It then sets parts of the belief as shown in (6)-(10) and adds it to the target-base in (11). Since a new belief has been added to the belief system, the **instructional-belief-update** procedure must be called to determine the new belief's own contradictions and dependent beliefs (12). Since **generate-derived** is part of **instructional-belief-update**, it illustrates the recursive nature of the parent procedure as well as itself. The belief generation process is performed until there is no inference rule that can be used to derive a new belief from an existent belief.

Example 6.6

Suppose the student previously held the belief

β_1 [((< duration 5) $\Rightarrow$ delusional-disorder],

and then after further interaction, the student later holds the belief

β_2 [((= duration 12) $\Rightarrow$ delusional-disorder].

If β_1 was labeled 'Correct' and was placed into $\mathcal{CBB}$, and β_2 is labeled 'Incorrect', the algorithm would delete β_1 from $\mathcal{CBB}$, and store β_2 into $\mathcal{ICBB}$, because

(< duration 5) = $\neg$(= duration 12)

because evaluation of (= duration 12) in the presence of (< duration 5) would yield a 'False'. If however we had

β_1 [((< duration 5) $\Rightarrow$ strong-moods] and

β_2 [((= duration 12) $\Rightarrow$ light-moods],

there is no contradiction since the preconditions and conclusions are mutually exclusive.

If no contradiction exists for β_2, it is stored $\mathcal{ICBB}$ anyway. This leads us to comment briefly about the initial state of CBB and ICBB, time which time neither belief base contains any belief. The first incoming correct and incorrect beliefs are stored in $\mathcal{CBB}$ and $\mathcal{ICBB}$, respectively, since no contradictions exist for them anyway. In this case, **find-contradiction** is still executed, only it yields a 'no-contradiction' contra-flag.

6.11 Complexity-analysis of algorithms

In this Section, we present estimates of the running times (in 'big-oh' notation) of the algorithms specified in this Chapter. Estimation techniques used here are taken from Aho et al. (1983), and are based on the numbers of inputs to the algorithms. Thus T(n) implies the running time of a specified algorithm, given an input size n. For the purpose of this discussion, we have used variable names as follows:

n: number of rules in KB, SL, or BK
m: number of symptoms for current case

p: number of paths in a specified set of paths
i: number of incomplete paths in current path trace
a: number of assumptions
c: number of compound beliefs
s: number of strategies
b: number of beliefs
g: number of incoming beliefs

The running time of Algorithm 6.1 is dependent on those of Algorithms 6.2 to 6.4, since it calls Algorithms 6.2 (which calls Algorithm 6.3) and 6.4. The running time of Algorithm 6.4 is the result of applying the *rule-of-sums* on lines (1) and (2), which is O(1)+O(number of symptoms) = O(m). Please recall that m represents number of symptoms in this Section.

Algorithm 6.2 is a recursive procedure which is also called by Algorithm 6.3. Since these two algorithms call each other, their running times must be estimated *in tandem*. We can first estimate the basic running time of Algorithm 6.2 by assuming some initial value for its input size, and then performing a recurrence calculation with different input sizes. The initial input size is not difficult to guess, since it is always stated explicitly before the procedure (***chain-rules***) is called. With an initial size dependent on the entire knowledge-base (n), we see that the worst-case time for (5) and (6) of Algorithm 6.3 is O(n). This gives us O(n) for the ***backtrack*** algorithm.

Returning to Algorithm 6.2, the running time will be O(n) multiplied by the longest of the sub-procedures in (3)-(4), (5)-(7), (8)-(9), and (10). This is O(n) * max(O(1), O(n), O(n), O(n)) = $O(n^2)$. This means that the running time of Algorithm 6.2 is a function of a square of the number of rules in the knowledge-base.

Now we can compute the running time for Algorithm 6.1, which is now the sum of O(1) for lines (1), (3) and (4), (11)-(13), O(m) for (5), $O(mn^2)$ for (6)-(9), and O(m) for (10). This would give us $O(mn^2)$. The while loop in line (2) would raise

this value by an integer factor, but essentially, the running time would remain a function of mn^2.

To obtain the running time of Algorithm 6.5, we must first compute that of 6.6, since the latter is called by the former. In Algorithm 6.6, lines (2)-(6) is of O(1), (7) -(9) is also of O(1). However, line (1) introduces the knowledge-base, which would mean a worst case of O(n). The result of applying the rule-of-products would therefore leave us with O(1) * O(n) = O(n) for this algorithm.

Back to Algorithm 6.5, line (10) is of O(n), which makes lines (8)-(10) of O(*in*), since it is performed as many times as the number of incomplete paths. Line (7) serves more as an explanatory line, and is actually redundant, since the major looping function is performed by line (8). From this it can be observed that the running time for lines (3)-(10) is max (O(1), O(n), O(in)) = O(in). When line (2) is included, the time updates to $O(in^2)$, for $i \geq 1$, for the entire algorithm. This actually reduces to a function of n^2.

The running time for Algorithm 6.7 is dependent on those of Algorithms 6.8 to 6.10. Algorithm 6.8 consists of elementary statements, and is of O(1). The same is true for Algorithm 6.9. Algorithm 6.10 is dependent on the number of assumptions (a), and is O(a) in lines (7) - (9) and (3). This makes the algorithm of O(a). From these estimates, we can see that the running time for Algorithm 6.7 is the larger of O(a), for lines (1) - (5), and O(c) for the compound beliefs of (6) - (8).

Algorithm 6.11 depends on the number of strategies, s, and number of beliefs, b. Lines (1) - (4) is of O(sb), while line (5) is of $O(s^2)$ for most sorting algorithms. We can therefore conclude that the running time of Algorithm 6.11

$$= \quad \{ \; sb, \text{ if } b > s$$

or { s^2, if $b \leq s$

For a large knowledge-base, we expect that s would normally be larger than b.

Algorithm 6.12 calls 6.13. From line (1), Algorithm 6.13 is of O(n) * O(1) of lines (2) - (8), which gives us O(n). Putting this into Algorithm 6.12, we get O(in) for lines (7) - (10), multiplied by O(n) of line (2). The rest is of O(1). Algorithm 6.12 is therefore of $O(in^2)$ for $i \geq 1$.

Algorithm 6.14 is of O(p) * O(1), for lines (1) and (2)-(10), respectively. This gives us O(p), which is the number of paths in BA. Algorithm 6.15 is the most complex algorithm, involving six loops. For lines (9) - (15), the running time would be the product of numbers of sub-beliefs and preconditions in the average paths in BA and BCONC. We are forced to make an estimate in this algorithm, since it is difficult to know precisely how many rules are involved in a path. However, by observing Figure 6.7, which is a good representation of a graph starting from the beginning, to somewhere near the end, of the knowledge-base. From this graph, we estimate that the path length is approximately 1/8th of the knowledge base, or 1/5th in the average case. Also, from observation of the knowledge-base rules, we estimate that the average number of preconditions in a rule would be 3. Using these figures, we obtain $O((3n/5)^2)$, which approximates to $O(n^2)$. The two outer loops in lines (3) and (4) introduce a product of the square of the numbers of paths in BA and BCONC. We could represent these in terms of p's, but in terms of n, the number of paths in BA and BCONC are, in practice, factors of the number of rules in the knowledge-base. This result of these outer loops would then give us an overall running time of $O(n^4)$, in the worst case, for Algorithm 6.15.

Algorithm 6.16 is a recursive algorithm which depends on Algorithms 6.17 - 6.21. Algorithm 6.19 is of O(1). Analysis of lines (1) - (8) of Algorithm 6.18 also produces a running time of O(1). The same result is obtained for Algorithm 6.17. Algorithm 6.20 is recursive, and calls Algorithm 6.16, its calling algorithm. Using an estimate for the initial value of the input size, the worst-case running time for 6.20 would be O(bg), since it depends on the number of incoming beliefs and the overall number of beliefs in the belief system. Algorithm 6.21 is of O(n) where n this time is the number of rules in the INFERENCE-RULEBASE. Algorithm 6.16 therefore depends mainly on the numbers of existent and incoming beliefs, b and g, respectively. Lines (3) - (6) are of O(bg), assuming that all existent beliefs are in ICBB (line (2)), and that all incoming beliefs are in CB of line (1). Introducing lines (1) and (2), that makes lines (1)-(12) of $O(bg)^2$. The same value is obtained from analysis of lines (13) - (25). This gives an overall running time of $O(bg)^2$ which points out a dependence on the numbers of existent and incoming beliefs, for the belief-revision algorithm.

6.12 Control flow of algorithms

Figure (6.10) shows the flow of control between the algorithms described in this Chapter. The algorithms are indicated in boxes, with rounded boxes representing recursive algorithms. Arrows show the execution order of algorithms, with dashed arrows indicating algorithms which are called by other algorithms. The major flow is from the EPS, which passes DD_{EPS} to the Knowledge-Finder, which passes computed strategies to the Response-Analyzer, which computes student's strategies and beliefs, and passes

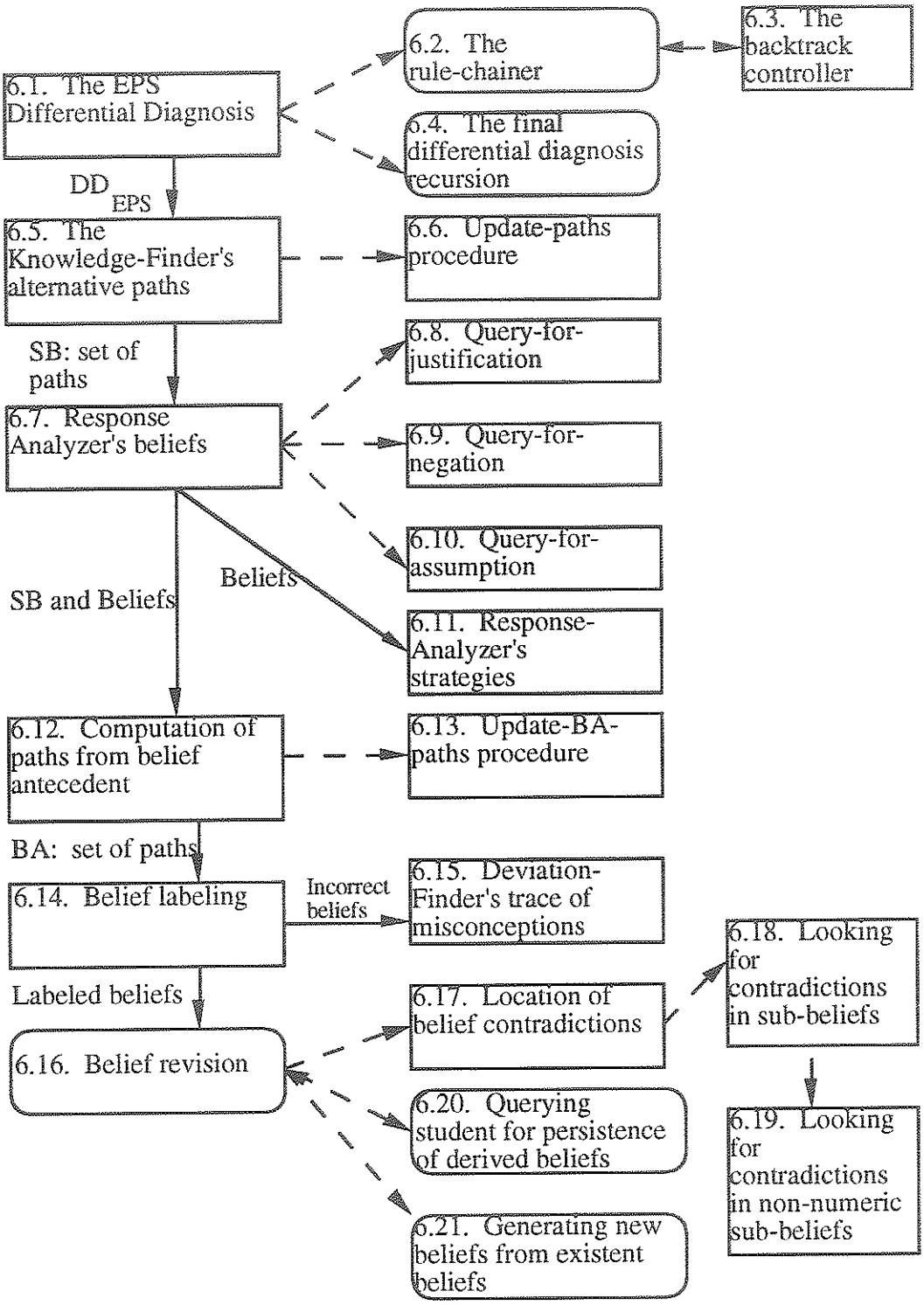


Figure 6.10: Control flow of algorithms

beliefs to the Deviation-Finder, which labels beliefs and computes misconceptions, and passes labeled beliefs to the Belief-Revision Unit. The flow within the Modeler stops at the Belief-Revision Unit, and when fitted into the overall Modeler structure (sees Figure 5.1 and 6.1), the next stage would be sending updated belief sets to the Tutor for further interaction with the student.

6.13 Summary

In this chapter we have described the Student Modeling Module of our model architecture in extensive detail. First we introduced the belief revision scheme we have developed for maintenance of a student's beliefs. The revision scheme is coherencist since the coherencist approach allows us to retain a derived belief in the belief system even when its justifier has been deleted. As we explained in Chapter 5, the coherencist approach conforms better to human cognitive behavior than the more restrictive foundationalist approach. Next we describe algorithms for the modeling process. Starting from the time a problem is presented to the student by the tutoring model, we specify algorithms for the Expert Problem Solver's solution process, the Knowledge Finder's computation of alternative paths to the EPS's solution, the Response Analyzer's query session with the student aimed at identifying student beliefs and solution strategies, the Deviation Finder's labeling of beliefs and identification of misconceptions using the knowledge base, and the Belief Revision Unit's maintenance of the overall belief system. In this section we summarize our approaches to the issues surrounding modeling and belief revision.

First, we identify a student's beliefs by means of a query process conducted by the Response Analyzer. Particularly, the **query-for-justification** procedure enables the Student Modeler to identify the reasons underlying a student's surface responses, which facilitates better modeling. We assume that the Response Analyzer's queries brings out

all of the student's beliefs used in his/her solution to the current problem. After a belief has been identified, we apply a list structure to represent the belief as a quintuple which uniquely identifies the belief and stores it as an implication along with its justifying beliefs and assumptions. The list data structure is easy to manipulate and has been used successfully by belief researchers to represent beliefs.

As explained in Chapter 5, the student's belief base differs from the Tutor's knowledge base, since the student may not believe every piece of knowledge in the knowledge base. Determination of a belief's correctness is a task performed by the Deviation Finder, using the knowledge base as the basis of evaluation. A student-belief is labeled correct if and only if there is some chain in the knowledge base that leads from the belief's antecedent to its conclusion. Checking for the correctness of intermediate conclusions is yet another step in the modeling process facilitated by the query process done by the Response Analyzer. The identification of deviations between the student's incorrect beliefs and the knowledge base is done by the Deviation Finder in a multi-step procedure. First it uses the precondition part of a student's incorrect belief to perform a forward chain through the knowledge base, to identify where the student's preconditions may have led him/her. Then it uses the belief's conclusion and performs a backward chain to determine what elements may have led to it. Finally it identifies similarities within the paths obtained from the two sets of chains, and draws reasonable inferences from comparisons. This is a very tedious process because the chains are listed exhaustively and the Deviation Finder tries to uncover as many explanations for the student's misconceptions as possible. This approach is unique to this model or design, and avoids using any overlay or buggy technique for reasons that have been discussed in previous sections and chapters.

When a belief has been identified and labeled, it must be sent to the pool of beliefs or belief base maintained for the student. Our model involves two labeled belief bases (one each for correct and incorrect beliefs) to enhance processing speed. In order to store a new belief, existent beliefs must be checked for contradiction. Since our beliefs are represented as implications, locating contradictions is performed by determining whether the conclusion parts of two beliefs conflict when their preconditions do not. This is the only contradiction condition in an implication-based formalism. Once it has determined the existence or nonexistence of a contradiction, the Belief Revision Unit proceeds to delete contradictions, store an incoming belief, obtain derivations using predefined inference rules, and continue the revision process recursively. In this model we have followed the coherencist school of thought, representative of which is the Dalal revision scheme. We have however shown that the Dalal belief revision scheme fails in a case of multiple incoming contradictions since it performs resolution of conflicts entirely on the originally unrevised belief base. Also, the Dalal scheme does not provide for data dependencies or derivations. Our model has improved on this scheme by performing recursive resolution of contradictions, and also by addressing the derivation of new beliefs as well as the recursive belief revision that follows from these derivations. We have also included the optimization technique of isolating non-contradictions from contradictions and performing a contraction process only on the contradictions, as was done by Boutilier and Goldszmidt. We have also demonstrated with multiple examples that our revision scheme satisfies the AGM postulates which constitute the contemporary standard of evaluation of coherencist schemes.

In a problem solving learning session in our prototype domain, we assume that the Tutor presents the student with symptoms and expects the student to return a differential diagnosis. At the same time the Expert Problem Solver performs a forward chain through the knowledge base to determine the correct diagnosis corresponding to the

given symptoms. This EPS's diagnosis (or DD_{EPS}) is used as a guide to determine the correctness of the student's own solution, without regards to problem-solving style or strategy. In order to promote strategy-independent student-solutions as discussed in Chapter 4, the Knowledge Finder uses the DD_{EPS} to perform a backward chain through the knowledge base, and lists exhaustively all paths that could lead to the same solution as the EPS's solution. These alternative paths assist the Response Analyzer in the determination of which strategy or strategies the student may be applying to his/her solution based on his/her responses to the Response Analyzer's queries. Candidate strategies are identified by the frequencies of the student's beliefs occurring in them.

After the belief revision process, the Modeler passes all beliefs (correct and incorrect), as well as missing beliefs, to the Tutor for further interaction with the student. As explained in our overview in Section 6.5, the further interaction would involve the Tutor addressing missing beliefs, and taking the student through material aimed at correcting incorrect beliefs. If at a later time the Tutor poses another problem to the student and the student produces a response, the modeling process will be activated all over again for modeling, that is, identification of student beliefs, computation of alternative paths, identification of misconceptions and missing beliefs, and more belief revision.

Overall, the components of the Student Modeling Module (or Modeler) work together to evaluate the student's responses, identify beliefs, promote individualized strategy and modeling, label beliefs, and maintain a consistent belief base. In the next chapter we summarize the entire thesis by highlighting our major contributions to research and directions for future work.

CHAPTER SEVEN: CONCLUSIONS

This chapter presents a summary of the foregoing chapters. We first highlight the differences between our model and some other intelligent tutors described earlier. Next we outline our contributions to research into Intelligent Tutoring Systems, and then discuss some extensions for future work.

7.1 Differences between our model and some other ITSs.

In this section we identify the differences between our model and the major intelligent tutoring systems described in Chapter 3. Where features are common in systems, we group them together.

a) One of the foremost differences between our model and current ITSs is in their use of an overlay model, while we use a strategy-independent model. These systems that use overlay models include GUIDON, WEST, WUMPUS, RBT, EEMT, SHERLOCK, and the STD/ITS.

b) We do not use a buggy model as in BUGGY or FITS, the fractions Tutor. We also do not use a mal-rule set since we evaluate the student on the basis of *deviation* from knowledge as opposed to matching to pre-programmed possible misconceptions.

GUIDON is the single intelligent tutoring system against which we can compare our work with respect to the medical domain. Therefore with specific reference to GUIDON, we identify the following difference:

c) GUIDON cannot perform constructive evaluation in the sense of understanding student misconceptions and offering explanations for them. We however achieve this feature by explaining misconceptions as deviations from rules. Our Deviation Finder identifies misconceptions by locating rules from the knowledge base from which the

student's beliefs deviate, thus allowing the tutoring model to explain misconceptions as deviations from rules rather than as conforming to some predefined mal-rules.

7.2 Contributions to knowledge

We summarize our model's contributions to knowledge as follows:

- integration of expert system and hypermedia technology into a general intelligent tutoring system framework. The hypermedia subsystem would facilitate browsing and representation of advanced graphics, sound, and video, while the expert system subsystem would enable the model to make intelligent inferences and to generate explanations.

- a path-independent student model which allows the student to apply any desired path to obtain a correct solution to a problem. This feature is made possible by the Knowledge Finder which uses the knowledge base to compute alternative paths to the same solution obtained by the Expert Problem Solver. These paths are then analyzed by the Response Analyzer which uses the student's beliefs to identify candidate paths the student may be applying. Path-independent modeling promotes more individualized tutor-student interaction since the overall system is better able to identify the student's solution style.

- identification of misconceptions using the double-chain analysis technique performed by the Deviation Finder. This differs from the traditional overlay and buggy models as was explained in Section 7.1. Each student-belief is first labeled correct or incorrect by determining whether there is at least one logical chain of inference in the knowledge base flowing from the belief's antecedent to the belief's conclusion. If a belief is identified as incorrect, the double-chain analysis is done by comparing forward-driven chains from the belief's antecedent with backward-driven chains to its conclusion, and

misconceptions are identified by comparing values of related keywords in the chains. This technique promotes identification of reasons underlying the student's incorrect responses.

- an improved coherencist belief revision scheme, $\mathcal{R}_{\mu}^{\Psi}$, based on the resolvent scheme discussed by Dalal (1988). $\mathcal{R}_{\mu}^{\Psi}$ revises a belief base Ψ with incoming beliefs μ. Unlike Dalal's scheme, $\mathcal{R}_{\mu}^{\Psi}$ is entirely set-theoretic, and does not fail in the face of multiple contradictions, since removal of contradictions is done recursively. It also incorporates derived beliefs which are dependent on the contradictions of incoming beliefs, and performs revision with derived beliefs recursively. We also showed that $\mathcal{R}_{\mu}^{\Psi}$ satisfies the AGM postulates which are the basis for evaluation of coherencist revision schemes. Maintaining a student's belief base using $\mathcal{R}_{\mu}^{\Psi}$ promotes minimal belief revision by changing the belief types of persistent derived beliefs, that is, by allowing the student to retain derived beliefs whose justifiers have been deleted if a query process indicates that the student still holds such beliefs.

7.3 Directions for future research

This final phase of this thesis concentrated only on the Modeler, which just one part of the overall model. Several aspects of the model therefore need to be addressed in future work. These aspects are discussed in this section.

a) first, we have only designed algorithms which could be used to obtain desired modeling results. These algorithms need to be implemented before other units of the overall tutoring model can be addressed. We have attempted to implement the very first two modeling phases described in Chapter 6: derivation of a correct diagnosis by the

Expert Problem Solver, and the computation of alternative solution strategies by the Knowledge Finder. Our programs yielded accurate results. That we obtained accurate results from these two phases is very significant because that means that the algorithms we have specified can be implemented since they are all inter-related and we employed the same design style and logical reasoning for them. The other phases however need to be implemented to obtain a working Modeler. This implementation can be done using Lisp since our belief structure is in list form and would therefore lend itself to manipulations with Lisp. In terms of ordering of implementation, the algorithms seem to flow in the order shown in Figure 6.9. We therefore suggest that the next series of implementation efforts should be directed at the Response Analyzer. This may involve designing a natural language interface that would drive the query sessions between the student and the Response Analyzer. Alternatively, we could use a highly structured command-oriented communication language to avoid extensive natural language processing. After the student's beliefs are successfully obtained from the Response Analyzer, implementation should next be directed at the Deviation Finder and then the Belief Revision Unit. Finally, it will be desirable to link the Modeler with the Tutoring Strategy Module through the belief bases, so the Tutor can pick up the updated student model and adapt tutoring material and complexity levels accordingly.

Other directions for future work on our model include the following:

b) programming the Tutor to include multiple tutoring styles for effective adaptation to the student. Refer to Joshua & Scuse (1995) for a discussion of multiple tutoring strategies. The Tutor should also be implemented to the level of domain-independence. This would involve development of templates for Tutor-student interactions, which would be filled in using the specific material from a chosen study domain.

c) development of a natural language communication interface, which would facilitate Tutor-student interactions.

d) development of the hypermedia components, which would involve storing actual learning material in an electronic textbook using a hypermedia tool (like HyperCard as suggested in Chapter 4), storing audio-visual clips in the audio/video units, and working with an instructor to design an appropriate Browser network for the subject area.

e) development of the Dynamic Questions Module with several schemata, as well as a full range-database indicating ranges of variable values, as well as the Variable-Relationship matrix. The presence of these would promote a better what-if query environment.

f) we notice that the estimated running times of the algorithms specified in Chapter 6 depend mostly on the Knowledge-Base, and are of $O(n^2)$, on the average, where n is the number of rules in the Knowledge-Base. Of particular note is the identification of misconceptions which is of $O(n^4)$. We believe that these running times can be reduced by applying heuristics to reduce the search spaces for these algorithms. One way of doing that is to include an OR-relationship among preconditions of rules, which strategy would reduce the size of the Knowledge-Base by close to 50%. Although this approach would complicate the implementation algorithms, we believe that it would visibly reduce the running times.

g) ultimately, it would be important to get all the model's units to interact as a running system, which would be the realization of the dream behind the design.

h) also, there is the possibility of using a neural network architecture to implement portions of the Student Modeling Module of this design. Neural networks offer more perceptive ability closer to the human brain than other artificial intelligence reasoning

inference techniques. In our opinion therefore, a neural-network-based Modeler might achieve more success at belief-identification than a procedural or mathematical model. However, the implementation of such a component is beyond the scope of this thesis.

7.4 Summary

In this thesis we have discussed several aspects of intelligent tutoring. We started by highlighting the current educational need: individualized instruction, which means the adaptation of tutoring content and style to the individual needs of a student. We then addressed different devices that have been used by educational technologists to promote individual instruction. These devices include television, radio, and computers. We then focused on computer-assisted instruction (CAI) which refers to the use of the computer as a medium of instruction, and discussed early (Ad-hoc Frame Oriented) systems which presented canned material and were not effective at individualization. Next we discussed intelligent tutoring systems (ITSs) or intelligent computer-assisted instruction (ICAI) systems which incorporate expert systems and provide more intelligent tutoring owing to the expert systems' inference ability. We examined several intelligent tutoring systems, which still had difficulty with the representation of graphics, personalized student modeling, and adaptation. We then presented a hypermedia-expert-system architecture to address these problems, with the hypermedia units facilitating multiple representation and the expert system units performing intelligent inference and generating detailed explanations using links to detailed material in the hypermedia units. Next we focused on the student modeling component of the module which constitutes the heart of any intelligent tutoring, system since adaptation of material and tutoring style would depend on intelligent student modeling. For this module we defined algorithms for path-independent student modeling (which allows a student to solve a problem using any desired solution strategy), identification of student's misconceptions using correct knowledge represented in a knowledge base, and a minimal coherencist

revision of the student's belief base. We have shown that this design is an improvement over several contemporary intelligent tutoring systems and we hope that it would help alleviate current student modeling problems and promoting individualized instruction.

APPENDIX A: OVERALL PROBLEM DEFINITION

We have graded the several aspects of our ICAI model according to levels of difficulty, as follows:

(Trivial)	(Not difficult)	(Difficult)	(Very Difficult)	(Extremely difficult)
[1]	[2]	[3]	[4]	[5]

Tasks that are outside the scope of this research are also graded as extremely difficult (for example, natural language interaction). Discussion of tasks follows.

1) Use of the Browser overview to identify and retrieve nodes to be taught. This can be viewed in two ways:

a) for uninterrupted learning of predetermined nodes, the student learns without any problem or need for branching. **Complexity (2).**

b) if there is need to branch off to remedial material as a result of the Student-Tutor interaction. The Tutor must decide what material to branch to, which is a function of the correct determination of the student's misconceptions. **Complexity (3).**

2) Administration of pre-test to ascertain student's prior knowledge of concepts.

a) Pre-testing involves the dynamic generation of questions at different levels of complexity. Templates in the Questions module are filled in with keywords and value *ranges* provided by the Tutor. **Complexity (2).**

b) A problem here is the determination of the degree of student knowledge of these pre-requisites. The number of correct answers to the pre-test may be used as a trivial measure of the degree of knowledge. This task is of **Complexity (2).** However, we intend to perform a delayed step-by-step evaluation of the student's solution, since a correct final answer may be obtained without a clear understanding

of intermediate steps and concepts. Step-by-step evaluation is of **Complexity (5)**. A student may not solve a problem using the same steps or approach as an expert, and the Tutor must possess the ability to interpret the student's goals from solutions.

c) Another problem is the identification of *which* pre-requisites or concepts are understood, and *which* are not, and to what degrees. Misconception identification is a tutoring perception problem of **Complexity (5)**, while the apportionment of credit or blame to a set of concepts, which is a pillar upon which subsequent pedagogy is performed, is of **Complexity (3)**, since it may be based mostly on weighted approximations. An inaccurate perception of student misconceptions leads to tutoring on irrelevant concepts and to poor student mastery of key material. This is a student modeling task, and is still a major research problem. Two approaches we have considered towards solving this problem are the use of a competitive neural network to produce a weighted set of misconceptions or the use of production rules plus certainty factors. Neural networks mimic the human brain and are better at perception problems than production systems. Neural networks have been used in diagnosis, expert-system and modeling research (Chen & Norcio 1992; Fu 1992; Fujita et al., 1992; Maren 1991; Papadourakis et al., 1992; Wang & el Ayeb 1992). However, the use of neural networks would further extend our work into yet another field in artificial intelligence. We shall therefore restricted ourselves to using production rules and certainty factors.

d) A third problem with pre-testing is the determination of what to do if pre-requisites are not understood, plus how many times to loop around clarification of pre-requisites before referring the student to a human instructor or textbook. We intend to solve this problem in a simple manner. A student can be allowed to backtrack to pre-requisites several generations (or levels) away from the current concept, until an

appropriate pre-requisite is found which the student understands. This follows the principle of adaptation, allowing the student to find his own level of cognitive competence before the Tutor proceeds with pedagogy. **Complexity (2).**

3) The Tutor will be programmed to assume a middle ground of complexity at the beginning of a learning session, and then move up or down the complexity scale based on student-Tutor interaction. Moving up the complexity scale, for instance, may involve asking more complicated questions, or providing less discussion details during learning.

a) We can trivially assume an average complexity level initially.

b) The major problem is how to determine whether the student is getting better or poorer at the concepts on the current complexity level: **Complexity (3).** Once it is determined how the student is faring, adjusting the complexity level would become a trivial task of **Complexity (2).** Determination of student's progress is tied to evaluation and modeling, and we can use responses to Tutor-questions, along with student queries and requests for clarification, to estimate overall progress on a concept.

4) The Fading principle: This is the process of providing detailed information at the beginning of learning, and gradually reducing details as the student's knowledge increases. Our Tutor will apply the fading principle, in order to properly adapt to the student's mastery levels. This is very important because the Tutor must not bore an average or expert student with excessive details. On the other hand, novice students can hardly progress without such details.

a) The problem here is the determination of the student's progress or otherwise, and the decision as to when to reduce the amount of information provided: **Complexity (4).** This is a perception and temporal decision, and must be taken at an appropriate

time. Particularly, pre-mature fading can be detrimental to learning, because a student might lose grip of certain key concepts as a consequence.

5) The system will generate questions dynamically, during interaction. The Dynamic Questions Module (DQM) contains resident formats/templates with which to compose questions when cued by the Tutor. The Tutor sends appropriate keywords and value ranges with which the templates are filled. Templates and value ranges are useful for the creation of "what-if" scenarios.

a) What we must consider here is the provision of appropriate templates for the problem domain, and the determination of which template(s) to choose for which problem. Appropriateness templates is salient because the Tutor needs to be equipped with the ability to generate challenging questions which will enhance learning, rather than trivial questions. We can provide templates that take care of as many known question formats as possible in the problem domain. This may however not be exhaustive since we cannot provide all possible question formats. We could also consider generating templates during interaction, if available templates are not suited to the current student. A major problem with pre-designed templates is that certain students need questions in certain unique formats, and may not be comfortable with available templates. Moreover, some students require that questions be asked them in progressive bits rather than whole, which would enable them to solve problems in phases. Templates can therefore be organized in bits which can be combined into larger chunks as the student's requirements change. Template design is of **Complexity (4).** Template generation is a more complex task, but we shall let it stay at level 4 for now.

6) Evaluation of student response:

a) This could be a pattern-matching or *overlay* process, in which the student's response is matched with the system's correct answer. An overlay scheme would be of **Complexity (2).** Our model will not employ the overlay scheme.

b) We believe that our Tutor should be capable of evaluating intermediate steps, which would assist it in the accurate determination of the student's beliefs, misconceptions and overall mastery level. As stated in 2b) above, however, intermediate evaluation is an extremely difficult task of **Complexity (5).** The Tutor must be able to keep track of the intermediate steps, and also to recognize the student's representations and solution style, which most times differ from the Tutor's approach, and are unique for each student.

7) Language of communication: this is closely related to the issue of representation mentioned in 6b) above. There must be a clear specification of what representations are permitted for the student. Unrestricted communication constructs would crash a system which has inadequate Natural Language Interaction-handling capabilities. Natural Language Interaction is outside the scope of this research, and so we shall allow for restricted language and clearly defined representations that will be adequate for meaningful student-Tutor interactions.

a) Options for communication language include the use of menus, commands, macros and keyword recognition, all of which we shall use for variety. This is a task of **Complexity (4).**

8) The Tutor will keep track of student response speed as a measure of mastery. This is a trivial task which can be solved using the implementation system clock.

9) The Tutor will query the student for reasons behind responses. It is easy to ask follow-up questions directly from a student's response.

a) The problem is how to understand the student's reasoning as the query process progresses. The issue of understanding is crucial, since the student can represent the same idea in ways different from the Tutor's. This is related to the problem of communication language as well as the identification of a student's goals in the problem-solution process. Goal identification is closely tied to step-by-step evaluation. The Tutor must be able to determine, from the student's solutions and responses, what goals the student has in mind, and what strategies he is trying to apply. Determination of goals is the opposite of planning. Planning is the decomposition of a goal into steps or actions, while goal identification is the synthesis of steps to recognize the parent goal. FITS (Nwana 1993) allows a student to specify intended goals. We, on the other hand, attempt to identify the student's goals without the student's assistance. This is a complex perception task of **Complexity (5)**, and involves the Tutor's knowledge of intricate steps involved in different solution strategies. Goals may overlap in the sense that two goals may involve some common steps in different sequences. The Tutor must therefore be able to recognize where one goal ends and another begins, as well as the ability to generate different goal-sets, if the student's steps suggest multiple goals. In the case of multiple goal identification, the Tutor must be able to determine which goal-set the student was actually pursuing, by follow-up questions during interaction.

10) The Tutor will allow the student to suspend learning, revise, or quit a course altogether. Suspension involves mostly book-keeping tasks of updating the student's study status records for use at resumption. Quitting a course may involve deleting a student's records from the Tutor's records. However, our Tutor will remember

students who have previously withdrawn from a course, by checking the *Quit* list each time a student registers for a course. This will enable the Tutor to determine more quickly an initial level of complexity for the student, based on the complexity level he was at before quitting. Revision involves recalling the path and nodes through which the student has traversed, in order to allow review. Sometimes though, the Tutor determines what needs to be revised. However, this is not a problem encountered during a normal revision process but rather as an offshoot of evaluation, which will be discussed in 11) below. Suspension, Revision and Quitting are all tasks of **Complexity (2)**.

11) The Tutor will administer post-tests at the end of topics, and give the student an option to review responses before evaluation.

a) Revision before evaluation involves collecting all responses and allowing the student to review or modify them, to ensure that such responses represent his final views before the Tutor proceeds to evaluate the responses. This is a mimic of the pen and paper environment, and is not an available feature in most CAI systems. This is a task of **Complexity (3)**.

b) The problem here is what happens after a post-test. The Tutor must determine, based on the student's performance, a belief about the student's overall understanding of the topic, as well as the identification of what key concepts have not been clearly understood by the student. The latter is a modeling problem, as outlined in 6) above. The next thing is to follow up identified misconceptions with further pedagogy or enforced revision, to ensure that the student has a comfortable grasp of the topic. Identification of misunderstood concepts is as explained in 2c). Revision is enforced here because the Tutor had obtained no notion that the concepts were not too difficult for the student. The concepts are therefore at the correct levels of complexity, but are not clearly understood. This differs from the

prerequisite-backtracking discussed for pre-tests in 2d). Since the student has actually studied these courses, the Tutor has only to ensure good mastery. This may also involve retrieving prerequisite or remedial material on the affected sub-topics. This may end up being a recursive process, in which the student does not understand a concept, nor its prerequisites, nor a prerequisite's prerequisites, and so on. Although we can say that this will rarely be the case, the Tutor will decide after two or three backtracks, to refer the student to a textbook or a human instructor. This is what we call an *instructor referral.* Records of instructor referrals are maintained by the Study-Management module. Overall post-test coordination, including the decision on what to do as a result of the student's performance, is of **Complexity (4).**

12) The Tutor will obtain correct solutions from the Expert Problem Solver and offers explanations, also from the EPS. It will use keywords to search for rule conclusions, thus locating the node addresses from rules in the knowledge base, and retrieving the required nodes from the electronic textbook: **Complexity (3).** The ability to present detailed information from the hypermedia unit allows the Tutor to offer better explanations than conventional expert-system-based tutors.

a) Sometimes the detailed explanations may not be obtainable from the rules. We shall therefore prepare to give explanations to the student on any keyword using an *index of keywords*, as is done in textbooks. This index would list all keywords in the electronic textbook and all the nodes in which each is mentioned. If a student requests an explanation of any keyword, the Tutor would retrieve the appropriate node addresses and present nodes to the student. The compilation of the index is of **Complexity (3)** while the presentation of the nodes is of **Complexity (2).**

13) Our Tutor will allow for a truly mixed-initiative interaction which permits the student to ask questions using any of the allowable communication keywords or commands. This goes beyond the work done in GUIDON, where the student is able to ask only "WHY" questions. This is a problem of **Complexity (5)**, because the Tutor has to be prepared for completely unexpected questions in any of the accepted formats, and must be able to explain its response to the student, who is permitted to ask 'what-if' questions.

a) A problem is the determination of what to do if the student asks a question that is completely outside the domain of the Tutor, or something that the Tutor does not quite understand. It is easy to handle Tutor ignorance by simply admitting that the concept is more than the Tutor can handle. One way however, of dealing with *extra-domain* questions is for the Tutor to tell the student that the question is **irrelevant** or **beyond the scope** of the concepts being discussed. This would involve a quick search for the keywords used in the student's question, from the Tutor's index, and generating a message if they are not found. This also takes care of the rare eventuality that a student would abuse the mixed-interaction facility by persistently asking distractive questions : **Complexity (2)**.

14) A very important feature to maintain is the history of interactions. This involves recording the beliefs and misconceptions displayed by the student, and updating the belief set when new ones are observed. The issue of belief revision has been discussed in great detail by Huang et al. (1991). Their work involved the maintenance of stereotype information as well as observations and deductions from student behavior. Update of their stereotype database was done by a combination of bottom-up constraint satisfaction and top-down default propagation, while maintenance of the deductive database was by use of an Assumption-based Truth Maintenance System. They however left out the process of identification of beliefs (that is the determination of the

student's conceptions and misconceptions), which is a major part of our research. We shall concentrate on the belief-identification process, and implement history maintenance at a more superficial level, at **Complexity (4)**.

A summary of the tasks defined above, grouped under complexity levels, follows.

COMPLEXITY (2)

- Coordination, retrieval and presentation of nodes during uninterrupted learning.
- Question generation using pre-designed templates
- determination of degree of knowledge by matching final answers with correct answers
- Assumption of middle ground of complexity, and subsequent adjustment.
- Backtracking through generations of concepts or pre-requisites to find an appropriate level of understanding.
- Coordination of learning suspension, revision and quitting.
- Lookup of keyword in index for unexpected queries and for explanation generation.

COMPLEXITY (3)

- Choice of which nodes to branch to for remedial material..
- Apportionment of credit / blame to a set of misunderstood concepts.
- Determination of student progress on a concept, via queries, student requests for help and responses.
- Locating and retrieving nodes for explanation purposes.
- Keyword-Index compilation.
- Collection of test responses for review before evaluation.

COMPLEXITY (4)

- Application of the fading principle, plus the decision on when to do so, after determination of student progress listed under Complexity (3) above.
- Template design, and possible generation during interaction.
- Communication language: macros, menus, commands and keyword identification.
- Coordination of a post-test and consequent interaction.
- Maintenance of interaction history.

COMPLEXITY (5)

- Delayed step-by-step (or intermediate work) evaluation.
- Determination of which concepts or pre-requisites are believed or misunderstood.
- Determination of student's solution goals and strategies, and the choice among multiple goals.
- Coordination of student initiatives and questions.

Complexity level (1) is reserved for trivial tasks. We shall concentrate efforts on the higher complexity tasks, with particular emphasis on Complexity levels 4 and 5. The expected contributions of our work to the knowledge world would be in the Complexity 5 tasks:

- student strategy identification,
- accurate identification of beliefs and misconceptions,
- delayed step-by-step evaluation, and
- coordination of student initiative.

APPENDIX B: IMPLEMENTATION APPROACHES ORDERED BY COMPLEXITY LEVELS

This appendix describes our implementation considerations and techniques applied towards the solution of the problems specified in Appendix A. Problem identifiers tally with those in Appendix A.

LEVEL 2

2.1. Coordination, retrieval and presentation of nodes during uninterrupted learning.

- Look up Tutoring-node (*T-node*) for node's address
- Retrieve from electronic textbook and present

2.2. Question generation using pre-designed templates.

- Receive
  - desired complexity level
  - keywords and ranges data types
- Lookup Range structures
  - randomly select a range from the range set
  - randomly select from appropriate string sets
- Randomly select appropriate template type
  - select complexity level
  - complete template with keywords and range values
  - present question

Note that the Questions module must not select any random value more than once in the same query sequence. Each selected value is marked (to avoid repetition) and all marked values are unmarked at the end of the sequence.

2.3. Response match: determination of degree of knowledge by matching final answers with correct answers.

- The system would maintain a count of correct student responses
- If student's answer matches correct response from EPS, increment count;
  - else decrement count
- In addition (or alternatively), system maintains certainty factors attached to student beliefs (*i.e. system's notion of the degree to which student believes each concept*). If student response is correct, increment certainty factor by a set value;
  - else decrement certainty factor

2.4. Assumption of middle ground of complexity, and subsequent adjustment.

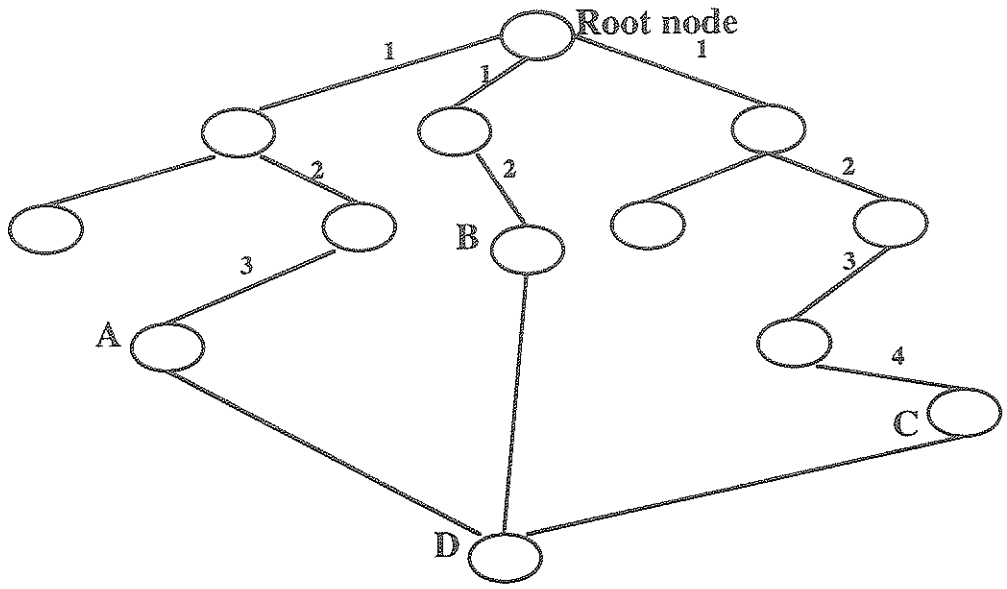
- There are grades of complexity with respect to questions and levels of conceptual information provided to student

- Initially contents of correct and incorrect belief sets = **null**
  - Assume there are five complexity levels; start at level 3
  - During interaction,
    - follow trend of degree of knowledge obtained in 2.3. above
    - increase / decrease complexity level as the measured degree of knowledge (or certainty factors) rises or falls, respectively
    - add new beliefs to belief sets
    - remove beliefs which have been contradicted by new ones

2.5. Backtracking through generations of concepts or pre-requisites to find an appropriate level of understanding.

- Each t-node has addresses of *pre-requisites* and *related concepts*
- If there is only one prerequisite, then it is located via its address, and presented

- If there are 2 or more prerequisites, then use browser overview to determine the least complex one. This will be done using a *depth* feature attached to each t-node. *Depth is a measure of nodal distance from the root concept.* The deeper a node is, the less complex it is.



- For instance, in the above network, nodes A, B and C are candidate prerequisites of node D, at depths 3,2 and 4, respectively.

- present the least complex pre-requisite (C in the above example). For equally likely pre-requisites, select randomly.

- Repeat location and presentation of prerequisites until the student gets to a comfortable level

- backtrack to a rejected prerequisite if path previously followed hits a dead end. This is a *reverse depth-first search* (since we are searching the network bottom-upwards)

- Note that link types differ:

- pre-requisite links - top-down or hierarchical
- related-concept links - adjacent, non-hierarchical

Only prerequisite links are followed during the backtracking process

2.6. Coordination of learning suspension, revision and quitting.

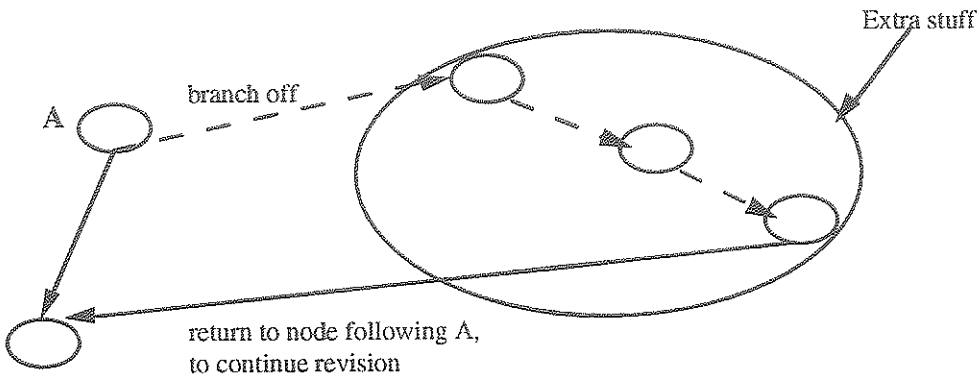
these are performed by the Study Management Module.

On suspension

- record current node, current belief set and associated certainty factors
- record all questions answered, problems solved, complexity level, and navigated path
- In short, the complete interaction history is committed to memory, on suspension.

For revision

- Recall the navigation path
- Ask for point at which revision is to start
- Re-present nodes
- Allow student to ask questions
  - retrieve correct answers from the EPS
  - revision must also involve pedagogy if the need arises
- Note that a major difference between normal learning and revision is that after branching from a node A, the student is obliged to return to the node following A, since he is undergoing a REVISION of the previously navigated path (see diagram below). This is not so in normal learning, in which the student is not obliged to study a node (and its descendants) that is currently too complex, but is allowed to branch to easier material.



For quitting

- Transfer student's records to the QUIT file but save
  - complexity level
  - current set of beliefs
  - set of major concepts studied
  - overall navigated path
- These data will assist the Tutor to adapt to the student more quickly, in case the student chooses to re-take the course at a later date.

For learning resumption

- Retrieved information saved at suspension
- Ask some revision questions
- If performance not good,
  - determine what concepts the student seems to have forgotten
  - backtrack to appropriate nodes (use previous path) and refresh student's memory
  - proceed with new material

2.7. Lookup of keywords in index for unexpected student queries and for explanation generation.

- The model will contain an alphabetically ordered index of technical keywords and terms, along with addresses of nodes in which they occur
- If student asks about a keyword and the knowledge-base cannot trace an exact or satisfactory explanation, the Tutor can use the index to retrieve nodes in which the terms occur
- The appropriate nodes will then be presented to the student in student's chosen order (or in sequential order of occurrence in index if student does not specify otherwise). The student may discontinue the lookup process whenever desired.

2.8. Coping with incomplete information in the event of unknown or unrecognized queries.

- If a keyword can neither be found in the knowledge base nor in the index, the Tutor would generate a message indicating that the desired information is not available in the system.
- The Tutor will save such terms in *unrecognized terms* table, for the instructor's attention.
- The unrecognized terms table can be useful for update purposes if the instructor sees the need to address the student's queries.
- In the case that any term requires system update, the instructor would perform the following tasks:
    - add relevant nodes to the electronic textbook
    - add relevant rules to the knowledge base

- adjust the hypermedia browser network links
- update the keyword index

- If an unrecognized term is not relevant to the study domain, the instructor would indicate that, plus the domain in which it is applicable. This information is stored in a node attached to the term or keyword

- At a subsequent learning session, the Tutor retrieves the nodes relevant to unrecognized terms, explaining the terms, or making recommendations for study of another domain affecting the keyword. This process ensures that the student's queries are responded to, at least to a reasonable extent.

- The foregoing implies that any specific keyword may be unrecognized to the Tutor a maximum of only one time. Once update or recommendations are inserted by the instructor, the Tutor can retrieve relevant responses for the student whenever such terms recur. The system thus grows (and the Tutor learns more) as new terms and keywords are introduced.

LEVEL 3

3.1. Identification of misconceptions: Note that it is very important that the Tutor uses follow-up questions to draw out all of the student's misconceptions at any stage, before interrupting with pedagogy. This enhances more effective teaching, and promotes a higher level of individualized problem-solving by the student.

Scenario: Tutor asks a question, and the student responds.

- Tutor asks a follow-up question on sub-concept (based on a student's response)
- Student responds
- Tutor finds rules relating to the sub-concept

- if the student's response is correct, mark the rule's belief as a positive belief

- If the student's response is incorrect, Tutor finds rules in the knowledge-base which the student is using incorrectly, that is rules which contain some relationships that the student is using, plus other entities not being used by the student. The beliefs represented in these rules are marked as partial beliefs held by the student.

- Repeat a) to d) until the student gives consistently correct responses, or until no new misconception is found relating to the local sub-concept. This second condition is relevant because there is no need to query a student beyond concepts that have been studied. It is natural for a student to display evidence of misconception of unfamiliar concepts.

- New beliefs discovered during this query process are used to update the belief base

3.2. Choice of which nodes to branch to for remedial material. This is a continuation of 3.1. above.

- For each misconception discovered do:
  - Present associated rules from the knowledge base
  - Locate addresses of associated electronic textbook nodes from rule and present
  - Lookup keyword index and present any node not yet presented in b). For multiple keyword relationships, perform an AND of nodal addresses and present any node not yet presented in b).

- The preceding is a basic explanation process. If concepts are not yet understood after explanations, then locate prerequisites of sub-concept and present. See 2.5 for prerequisite location.

- Note that nodes are identified as either major sub-concepts or constituent nodes. If a node relating to a misconception is a major sub-concept, then a series of nodes would be presented by the Tutor. Otherwise, the Tutor simply presents single relevant nodes.

3.3 Determination of student progress on a concept, via queries, student requests for help, and student responses.

- A major part of this has been tackled in 3.1 and 3.2.
- When a student requests for help:
    - mark the help sub-concept temporarily as a misconception
    - retrieve rules and nodes associated with requested sub-concept
    - perform pedagogy if necessary. Also, use follow-up questions to locate further hidden misconceptions associated with the key help sub-concept

3.4. Locating and pulling out nodes for explanation: See first part of 3.2 above.

3.5. Keyword index compilation.

- Index compilation must not be performed manually.

- The model will contain a list of keywords and technical terms.

- Using the list, and for each term or keyword, the Tutor does the following:
    - perform search of nodes
    - record address of every node in which keyword occurs
    - often, a keyword is the subject of a major heading, and would likely occur

in most nodes in that section. For such cases, record the major heading's start address.

- Then for all major headings, there is a list of constituent nodes. This separate detail is necessary for the performance of AND or OR operations for locating nodes for explanations involving multiple terms or keywords.

3.6. Collection of test responses for review before evaluation: This is a buffer function.

- Receive and save all responses

- For each response,
  - display former response and ask for revisions
  - re-save

- Ask student to indicate when done with review (system may allow say 2 overall review cycles)

- Activate the Tutor for evaluation of individual responses

- Tutor-Student interaction must be done during the evaluation process. The Tutor must query for reasoning behind correct responses, and also perform diagnostic queries for misconception identification in the case of incorrect responses. Since student responses are saved, the Tutor does not forget or lose any response, but can take time to address each student response as is necessary.

- Note that the key issues here are that
  - the student is allowed to complete solutions to personal satisfaction, and without interruption too

- the Tutor must not rush evaluation, but must tackle each response, and query student for reasoning, in order to perform relevant pedagogy if necessary.

LEVEL 4

4.1. Application of the fading principle, plus decision on when to do so, after determination of student progress (see 3.3.)

- Nodal information is organized in hierarchical levels of detail:
  - ABSTRACT level (lowest level)
  - MEDIUM level
  - DETAILED level (highest level)
- Each level expands on the information provided in the preceding level
- The abstract, medium and detailed levels are appropriate for the advanced, average and novice students respectively.
- How to determine the appropriate level for a learner:
  - It is easier to detect when a student is getting too little information (student would ask for explanations) than when he is getting too much information (student may feel irritated but say nothing).
  - It is therefore better to start at the advanced / abstract level, and adjust upwards as needed
  - During interaction, as the student's responses become consistently more accurate, adjust detail level downwards until the student starts asking for more explanations.

•• This is a dynamic adaptation process.

4.2. Template design and possible generation during interaction.

- Ideally, templates should be generated dynamically by the Tutor, but this would be in a truly natural-language-driven generator system. In the absence of such a driver, we must make do with an inexhaustive but supposedly sufficient set of templates.

- Templates in this system are organized in levels of complexity, from simple definition templates, to complex reasoning templates, involving one, two or more variables.

- Template types are domain-independent. They are however completed with domain-dependent keywords, value ranges and random string selections.

Template examples follow:

Single-variable template for variable X

[*string*] X ?

The string indicated above is a required string, which mostly determines the nature of the question posed to the student. For instance

- What is X ? (a definition question)
- What do you understand by X ? (a definition or comprehension question)
- Compute X (a problem-solving question)
- Why X ? (query for reasoning)
- How did you arrive at X ? (query for reasoning)
- Can you explain: X ? (query for reasoning)

Double-variable template for X and Y

[*string-1*] X [*string-2*] Y

Again, the strings indicated in brackets are required, and determine the nature of the question. For instance,

- Is there a relationship between X and Y ? (query for relationship understanding)
- How does X differ from Y ? (query for comparison)
- Does X affect Y ? (query for relationship understanding)
- Can you compute X given Y ? (problem-solving query)
- Why do you think X may imply the {presence, absence} of Y ? (query for reasoning)

Multiple-variable example

- So you think Z results from X and Y ? (query for reasoning involving 3 variables)

Note that the template unit grows as the system.

- Each template includes number of variables

- For each query type, there are several complexity levels, and for each complexity level, there are sets of query string forms.

- There are three classification strategies:
    - by number of keywords or variables
    - by query type
    - by complexity level

Organizational options are shown in the table below:

Major classifier	Second level	Third level
Number of keywords	Complexity level / query type	Query type / complexity level
Query type	Complexity level / number of keywords	Number of keywords / complexity level
Complexity level	number of keywords / query type	Query type / number of keywords

- Classification strategy greatly affects the efficiency of query selection during learning. For example, suppose queries are classified by query type (e.g. comprehension, relationship, reasoning, definition, etc.). Suppose there are two variables in a certain query, and the Tutor wants to test for comprehension, relationship and reasoning. First the Tutor would access the comprehension template class, and select queries. Next, the Tutor accesses the relationship template class, and then the reasoning template class. There is thus a jump from one template class to another, and probably back and forth. The same process applies to classification by complexity level, especially since the Tutor must adapt dynamically to the student's understanding. Classification by the number of variables or keywords includes all the appropriate query types and complexity levels. It is therefore the most preferred classification strategy, since selection of different forms can be performed within the same template class.

4.3. Communication language: macros, menus, commands and keyword identification.

- Used for interaction and request specification during a learning session
- grouped under synonyms
- also grouped in novice / advanced student formats (not all)
- For each communication code, the system contains:

- code
- meaning / function
- synonyms
- syntax

Macro / Command Examples

- REGISTER {for course registration}
- SUSPEND {for learning session suspension}
- REVISE {must include start location for revision}
- QUIT {for quitting a course altogether}
- WHY {request for explanation}
- WHAT IS {request for explanation}
- EXPLAIN {request for explanation}

- Synonyms for EXPLAIN include CLARIFY, HOW_COME, and HELP [X]

- Function keys and control keys can be programmed as intermediate or advanced code keys. For example, F1 or ^R for intermediate and advanced inputs for REGISTER, respectively.

- Menus: provided for query and operational formats. All commands and macros are also available in menus, and there are different menus for login, logout, explanation requests and other functional activities. An example follows:

- For explanations

  - WHY
  - HOW_COME
  - EXPLAIN
  - NOT_CLEAR
  - CLARIFY

- Technical language

  - EFFECT_OF
  - RESULTS_IN
  - IMPLIES
  - INCLUDES
  - DIFFERS_FROM

Communication codes and syntax will grow with the system.

4.4 Coordination of post-tests and subsequent interaction.

- Collect test responses.
- Allow student to review before evaluation. Save final responses
- For each response
    - query for reasoning (see 3.1.)
    - mark any incorrectly answered or incorrectly reasoned-out problems
    - diagnose or identify misconceptions (see 3.1.)
    - perform remedial pedagogy if necessary (see 3.2)
    - dynamically modify belief sets
- Repeat test on previously marked problems. If there is substantial progress, then okay
- Repeat testing, diagnosis and pedagogy until the student's misconceptions are cleared
- Note that the student may suspend the current post-test topic if desiring to study an unrelated or non-subsequent topic. However, the student cannot progress to new topics for which current post-test topic is a pre-requisite

4.5. Maintenance of interaction history: This is the task of the Study Management Module and includes keeping track of

- current complexity level
- path of navigated nodes
- concepts learned
- degree of knowledge
- belief sets

All of these have been addressed in Complexity level 3.

LEVEL 5

5.1. Step-by-step (or intermediate work) evaluation: This has been discussed in 3.6 and 4.4.

5.2. Determination of which concepts or pre-requisites. See 3.1. to 3.3.

5.3. Coordination of student initiatives and questions. See 3.3. and 4.3.

5.4. Determination of student's goals and strategies, and the choice among multiple goals. Identification of strategies will be done by matching the student's beliefs against a strategy-base computed using the Expert Problem-Solver's final response as a guide. We can attempt to identify goals during problem-solving as described below.

- Each rule has preconditions and postconditions. The preconditions are the rule antecedents while the postconditions are the rule conclusions. For instance for the rule

  IF X AND Y THEN Z,

  the preconditions are X and Y while the postcondition is Z.

- There will be two lists for goal determination: GOALS LIST and PRECONDITIONS LIST. When a student requests data, the requested data is added to the PRECONDITIONS LIST. Also, initial information provided to the student are preconditions.

- When a rule is fired whose antecedent exist in the PRECONDITIONS list, the rule's conclusion is added to the GOALS list. It is important not to delete existent

preconditions because they may be required for the firing of more rules whose conclusions are also added to the GOALS list.

- When a rule is fired whose antecedent involves an entity in the GOALS list (say Z), that entity is deleted from the GOALS list and transferred to the PRECONDITIONS list (and marked as a subgoal) while the new larger conclusion is added to the GOALS list. For instance

PRECONDITIONS	GOALS
X	Z (deleted)
Y	W(new, involving Z)
Z(added; marked as subgoal)	

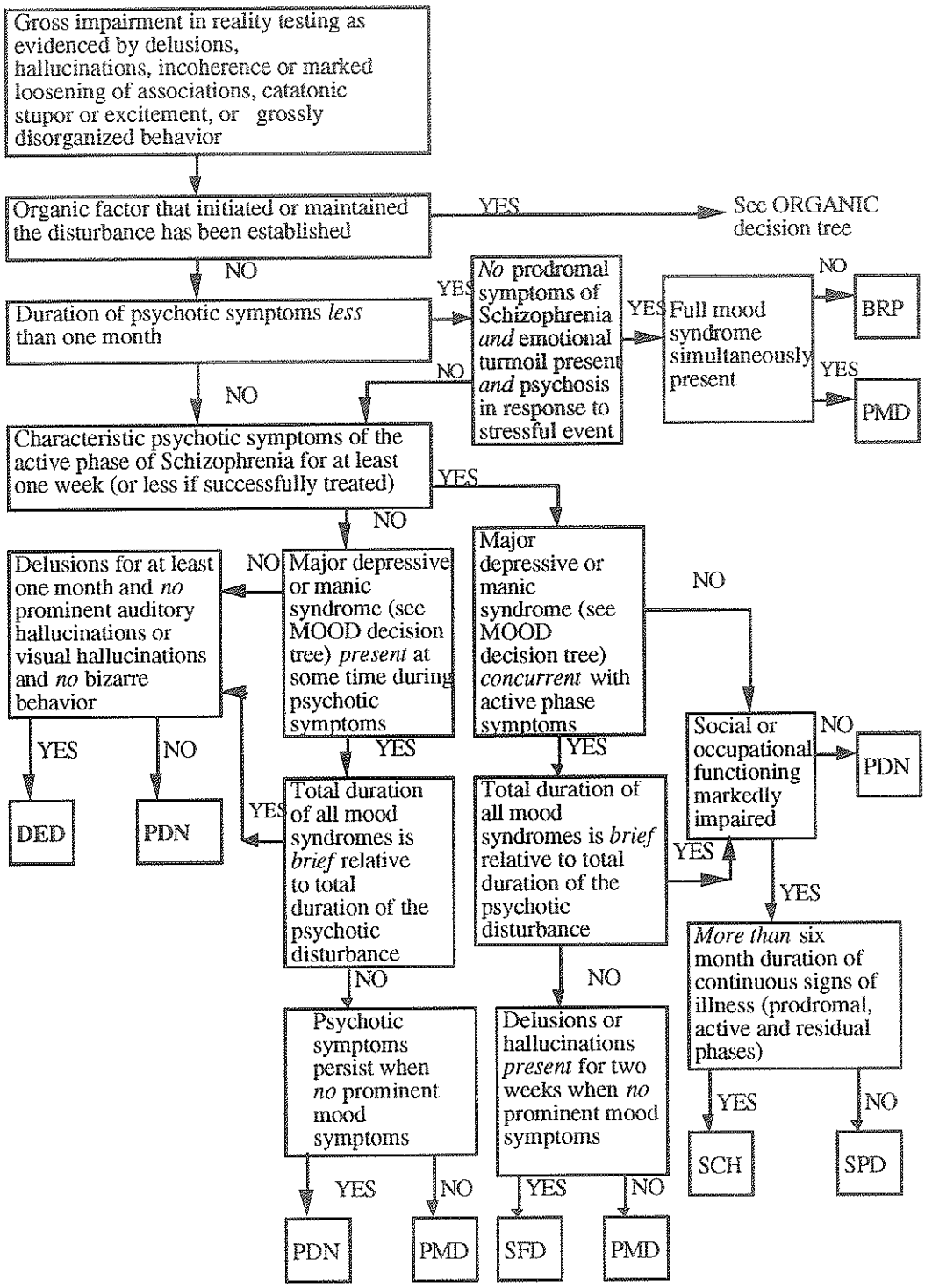
- There is thus a continuous update of the preconditions and goals until the student arrives at a solution. The GOALS list gives a clear idea about how the student's solution was derived.

- Goal determination will also assist the Tutor in the measurement of the student's plan optimality. This can be derived from the number of preconditions which do not appear as antecedents in the student's final GOALS list. The Tutor can therefore make solution strategy recommendations based on this information.

APPENDIX C: Differential Diagnosis of Psychotic Symptoms
(Based on the DSM-III-R Manual)

ABBREVIATIONS

BRP: Brief Reactive Psychosis
DED: Delusional Disorder
PDN: Psychotic Disorder Nos
PMD: Psychotic Mood Disorder
SCH: Schizophrenia
SFD: Schizoaffective Disorder
SPD: Schizophreniform Disorder


Gross impairment in reality testing as evidenced by delusions, hallucinations, incoherence or marked loosening of associations, catatonic stupor or excitement, or grossly disorganized behavior
Organic factor that initiated or maintained the disturbance has been established
YES
See ORGANIC decision tree
NO
Duration of psychotic symptoms *less* than one month
YES
No prodromal symptoms of Schizophrenia *and* emotional turmoil present *and* psychosis in response to stressful event
YES
Full mood syndrome simultaneously present
NO
BRP
YES
PMD
NO
NO
Characteristic psychotic symptoms of the active phase of Schizophrenia for at least one week (or less if successfully treated)
YES
NO
Delusions for at least one month and *no* prominent auditory hallucinations or visual hallucinations and *no* bizarre behavior
NO
Major depressive or manic syndrome (see MOOD decision tree) *present* at some time during psychotic symptoms
Major depressive or manic syndrome (see MOOD decision tree) *concurrent* with active phase symptoms
NO
YES
NO
YES
YES
DED
PDN
YES
Total duration of all mood syndromes is *brief* relative to total duration of the psychotic disturbance
Total duration of all mood syndromes is *brief* relative to total duration of the psychotic disturbance
Social or occupational functioning markedly impaired
NO
PDN
YES
YES
NO
Psychotic symptoms persist when *no* prominent mood symptoms
Delusions or hallucinations *present* for two weeks when *no* prominent mood symptoms
More than six month duration of continuous signs of illness (prodromal, active and residual phases)
YES
NO
YES
NO
YES
NO
PDN
PMD
SFD
PMD
SCH
SPD

APPENDIX D: KNOWLEDGE-BASE RULES

As mentioned in the main thesis, our prototype implementation was done in the domain of psychiatry. We used the DSM-III-R (American Psychiatric Association 1987; Kaplan & Sadock 1985) structure as the basis of our diagnostic algorithms and knowledge-base information. The DSM-III-R is a very helpful instrument in differential diagnosis of mental disorders, since it highlights the common symptoms among similar disorders, as well as detailed differentiation among disorders with the introduction of more symptoms. What we have done is to transcribe the DSM-III-R structures into expert system knowledge, for use in psychiatric instruction. In this section we outline the rules obtained from a single flow structure (a directed acyclic graph) based on the DSM-III-R: the Differential Diagnosis of Psychotic Symptoms (shown in Appendix C). Note that DURATION refers to symptom-duration, which is measured in weeks. This appendix shows our knowledge-base rules in Lisp form.

```
(defun setup-knowledge-base()       ; construct the knowledge-base

 (setq *KNOWLEDGE-BASE*
    '((rule-01  ((equal delusions T))
            (psychotic-symptoms-positive)
            (delusional-disorder
                        psychotic-disorder-nos
                        psychotic-mood-disorder
                        brief-reactive-psychosis
                        schizophrenia
                        schizoaffective-disorder
                        schizophreniform-disorder
                        organic-mental-disorder))

      (rule-02  ((equal incoherence T))
            (psychotic-symptoms-positive)
            (delusional-disorder
                        psychotic-disorder-nos
                        psychotic-mood-disorder
                        brief-reactive-psychosis
                        schizophrenia
                        schizoaffective-disorder
                        schizophreniform-disorder
```

```
                organic-mental-disorder))

(rule-03 ((equal loosened-associations T))
      (psychotic-symptoms-positive)
      (delusional-disorder
            psychotic-disorder-nos
            psychotic-mood-disorder
            brief-reactive-psychosis
            schizophrenia
            schizoaffective-disorder
            schizophreniform-disorder
            organic-mental-disorder))

(rule-04 ((equal catatonic-stupor T))
      (psychotic-symptoms-positive)
      (delusional-disorder
            psychotic-disorder-nos
            psychotic-mood-disorder
            brief-reactive-psychosis
            schizophrenia
            schizoaffective-disorder
            schizophreniform-disorder
            organic-mental-disorder))

(rule-05 ((equal excitement T))
      (psychotic-symptoms-positive)
      (delusional-disorder
            psychotic-disorder-nos
            psychotic-mood-disorder
            brief-reactive-psychosis
            schizophrenia
            schizoaffective-disorder
            schizophreniform-disorder
            organic-mental-disorder))

(rule-06 ((equal hallucinations T))
      (psychotic-symptoms-positive)
      (delusional-disorder
            psychotic-disorder-nos
            psychotic-mood-disorder
            brief-reactive-psychosis
            schizophrenia
            schizoaffective-disorder
            schizophreniform-disorder
            organic-mental-disorder))

(rule-07 ((equal behavior 'grossly-disorganized))
      (psychotic-symptoms-positive)
      (delusional-disorder
            psychotic-disorder-nos
            psychotic-mood-disorder
            brief-reactive-psychosis
```

```
                    schizophrenia
                    schizoaffective-disorder
                    schizophreniform-disorder
                    organic-mental-disorder))

(rule-08  ((equal psychotic-symptoms-positive T)
          (equal organic-factor T))
         (organic-mental-disorder)
         (organic-mental-disorder))

(rule-09  ((equal psychotic-symptoms-positive T)
          (equal organic-factor nil))
          (non-organic-disorder)
          (delusional-disorder
                    psychotic-disorder-nos
                    psychotic-mood-disorder
                    brief-reactive-psychosis
                    schizophrenia
                    schizoaffective-disorder
                    schizophreniform-disorder))

(rule-10  ((equal non-organic-disorder T)
          (< duration 4)
          (equal schizophrenic-prodromal-symptoms nil)
          (equal emotional-turmoil T)
          (equal stress-response 'psychotic))
         (light-moods)
         (psychotic-mood-disorder
                    brief-reactive-psychosis))

(rule-11  ((equal non-organic-disorder T)
          (>= duration 4)
          (equal schizophrenic-prodromal-symptoms T))
         (strong-moods)
         (delusional-disorder
                    psychotic-disorder-nos
                    psychotic-mood-disorder
                    schizophrenia
                    schizoaffective-disorder
                    schizophreniform-disorder))

(rule-12  ((equal non-organic-disorder T)
          (>= duration 4)
          (equal emotional-turmoil nil))
         (strong-moods)
         (delusional-disorder
                    psychotic-disorder-nos
                    psychotic-mood-disorder
                    schizophrenia
                    schizoaffective-disorder
                    schizophreniform-disorder))
```

```
(rule-13  ((equal non-organic-disorder T)
        (>= duration 4)
        (equal stress-response (not 'psychotic)))
       (strong-moods)
       (delusional-disorder
                psychotic-disorder-nos
                psychotic-mood-disorder
                schizophrenia
                schizoaffective-disorder
                schizophreniform-disorder))

(rule-14 ((equal light-moods T)
       (equal full-mood-syndrome T))
      (psychotic-mood-disorder)
      (psychotic-mood-disorder))

(rule-15 ((equal light-moods T)
       (equal full-mood-syndrome nil))
      (brief-reactive-psychosis)
      (brief-reactive-psychosis))

(rule-16  ((equal strong-moods T)
        (equal schizophrenic-prodromal-symptoms T ))
       (schiz-group-1)
       (psychotic-disorder-nos
                psychotic-mood-disorder
                schizophrenia
                schizoaffective-disorder
                schizophreniform-disorder))

(rule-17 ((equal strong-moods T)
       (equal schizophrenic-prodromal-symptoms nil))
      (psycho-delusions-1)
      (delusional-disorder
                psychotic-disorder-nos
                psychotic-mood-disorder))

;     Rule-18 deleted from here

(rule-19  ((equal schiz-group-1 T)
       (equal manic-syndrome nil))
       (schiz-group-2)
       (psychotic-disorder-nos
                schizophrenia
                schizophreniform-disorder))

(rule-20  ((equal schiz-group-1 T)
       (< mood-duration  psychotic-disturbance-duration))
       (schiz-group-2)
       (psychotic-disorder-nos
                schizophrenia
                schizophreniform-disorder))
```

```
(rule-21  ((equal schiz-group-1 T)
        (equal manic-syndrome T)
        (>= mood-duration psychotic-disturbance-duration))
        (schiz-group-3)
        (psychotic-mood-disorder
                schizoaffective-disorder))

(rule-22  ((equal schiz-group-2 T)
        (equal social-effect 'impaired))
        (schiz-group-4)
        (schizophrenia
                schizophreniform-disorder))

 (rule-23 ((equal schiz-group-2 T)
       (equal social-effect (not 'impaired)))
       (psychotic-disorder-nos)
       (psychotic-disorder-nos))

(rule-24  ((equal schiz-group-4 T)
        (> continuous-par-time 24))
        (schizophrenia)
        (schizophrenia))

(rule-25  ((equal schiz-group-4 T)
        (<= continuous-par-time 24))
        (schizophreniform-disorder)
        (schizophreniform-disorder))

(rule-26  ((equal schiz-group-3 T)
        (equal hallucinations T)
        (equal mood-syndrome nil))
       (schizoaffective-disorder)
       (schizoaffective-disorder))

(rule-27  ((equal schiz-group-3 T)
        (equal delusions T)
        (equal mood-syndrome nil))
       (schizoaffective-disorder)
       (schizoaffective-disorder))

(rule-28  ((equal schiz-group-3 T)
        (equal mood-syndrome T))
       (psychotic-mood-disorder)
       (psychotic-mood-disorder))

(rule-29 ((equal schiz-group-3 T)
        (equal hallucinations nil)
        (equal delusions nil)
        (equal mood-syndrome T))
        (psychotic-mood-disorder)
        (psychotic-mood-disorder))
```

```
(rule-30  ((equal psycho-delusions-1 T)
        (equal manic-syndrome T)
        (>= mood-duration psychotic-disturbance-duration))
        (psycho-delusions-2)
        (psychotic-mood-disorder
                psychotic-disorder-nos))

(rule-31  ((equal psycho-delusions-1 T)
        (equal manic-syndrome nil))
        (psycho-delusions-3)
        (delusional-disorder
                psychotic-disorder-nos))

(rule-32  ((equal psycho-delusions-1 T)
        (< mood-duration psychotic-disturbance-duration))
        (psycho-delusions-3)
        (delusional-disorder
                psychotic-disorder-nos))

(rule-33  ((equal psycho-delusions-2 T)
        (equal psychotic-symptoms T)
        (equal mood-symptoms nil))
        (psychotic-disorder-nos)
        (psychotic-disorder-nos))

(rule-34  ((equal psycho-delusions-2 T)
        (equal psychotic-symptoms nil))
        (psychotic-mood-disorder)
        (psychotic-mood-disorder))

(rule-35  ((equal psycho-delusions-2 T)
        (equal mood-symptoms T))
        (psychotic-mood-disorder)
        (psychotic-mood-disorder))

(rule-36  ((equal psycho-delusions-3 T)
        (equal delusions T)
        (>= duration 4)
        (equal auditory-hallucinations nil))
        (delusional-disorder)
        (delusional-disorder))

(rule-37  ((equal psycho-delusions-3 T)
        (equal behavior (not 'bizarre))
        (equal visual-hallucinations T))
        (delusional-disorder)
        (delusional-disorder))

(rule-38  ((equal psycho-delusions-3 T)
        (equal delusions nil))
        (psychotic-disorder-nos)
        (psychotic-disorder-nos))
```

```
(rule-39  ((equal psycho-delusions-3 T)
        (equal auditory-hallucinations T))
        (psychotic-disorder-nos)
        (psychotic-disorder-nos))

(rule-40  ((equal psycho-delusions-3 T)
        (equal visual-hallucinations nil))
        (psychotic-disorder-nos)
        (psychotic-disorder-nos))

(rule-41   ((equal psycho-delusions-3 T)
         (equal behavior 'bizarre))
         (psychotic-disorder-nos)
         (psychotic-disorder-nos)))))
```

REFERENCES

Abelson, R. P. (1979). Differences between belief and knowledge systems. In (eds.) Schank, R.; Collins, A. and Charniak, E. *Cognitive Science*, 3. pp. 355-366. New Jersey: Ablex Publishing Corp.

Abelson, R. P. & Reich, C. M. (1969). Implicational molecules: A method for extracting meaning from input sentences. *Proc. 1st Joint Conf. on Artificial Intelligence (IJCAI).* pp. 641-647.

Aho, A. V.; Hopcroft, J. E. & Ullman, J. D. (1983). *Data Structures and Algorithms.* Reading, Massachusetts: Addison-Wesley. 427p.

Akscyn, R.M.; McCracken, D.L. & Yoder, E.A. (1988). KMS: a distributed hypermedia system for managing knowledge in organizations. In *Comm. ACM, 31(7),* July 1988. pp. 820-835.

Alchourrón, C. E., Gardenförs, P. & Makinson, D. (1985). On the logic of theory change: partial meet contraction and revision functions. *The Journal of Symbolic Logic*, 50(2). pp. 510-530.

Alessi, S. M. & Trollip, S. R. (1991). *Computer-based instruction: methods and development.* 2nd Edition. New Jersey: Prentice-Hall. 512p.

Allen, M. W. (1975). Computer-managed instruction: A definitive design. *Proceedings of the IFIP 2nd World Conf. in Education, Paris*, pp. 115-122.

Ambrose, D. W. (1991). The effects of hypermedia in learning: a literature review. In *Educational technology,* 31(12). pp. 51-55.

Anderson, J. R. (1980). *Cognitive Psychology and its Implications.* San Francisco: W. H. Freeman and Company. 503p.

Anderson, J. R. (1984). Cognitive psychology and intelligent tutoring. In *Proc. 6th Annual Conf. of the Cognitive Science Society, Boulder, Colo.,* pp. 37-43.

Anderson, C.W. & Roth, K.J. (1989). Teaching for meaningful and self-regulated learning of science. In: (ed.) Brophy, J., *Advances in Research on Teaching*(1), pp. 265-309.

Angibeaud, J. R. P. (1979). Limits in computer-aided education. *Proceedings, International Conf. on Teleinformatics, Paris.* 99-102.

Arocha, J. F.; Patel, V. L. & Patel, Y. C. (1993). Hypothesis generation and the coordination of theory and evidence in novice diagnostic reasoning. In (ed..) Beck, J. R. *Medical Decision Making,* 13(3). pp. 198-211. Philadelphia: Hanley & Belfus.

Baker, E. & Quellmalz, E. (1972). *Research-based techniques for instructional design.* US. National Center for Educational Research and Development. 307p.

Baril, D.; Greer, J. E. & McCalla, G. I. (1991). Student modeling with confluences. In *Proc. 9th National Conf. on Artificial Intelligence(AAAI-91),* 1. pp. 43-48. Menlo Park: MIT Press.

Barr, A. & Feigenbaum, E. A. (1982). (Eds.), *The handbook of artificial intelligence*, 2, California: William Kaufman Inc.

Bergeron, A & Paquette, G. (1988). Discovery environments and intelligent learning tools. In (eds.) Frasson, C. and Gauthier, G. *Intelligent tutoring systems: at the crossroads of artificial intelligence and education.* pp. 34-55. New Jersey: Ablex Publishing Corp.

Bertin, A.; Buciol, F. & Lanza, C. (1992). Concepts of didactics in intelligent training systems. *First IEE Int'l Conf. on intelligent systems. Edinburgh.* pp. 257-262.

Billings, D. M. & Cobb, K. L. (1992). Effects of learning style preferences, attitude and GPA on learner achievement using computer assisted interactive videodisc instruction. *J. of computer-based instruction,* 19(1). pp. 12-16.

Blumenfeld, P.C. (1992). The task and the teacher: enhancing student thoughtfulness in science. In: (ed.) Brophy, J., *Advances in Research on Teaching*(3), pp. 81-114.

Boone, R & Higgins, K. (1991). Hypertext / Hypermedia information presentation: developing a HyperCard template. *Educational technology,* 31(2). pp. 21-30.

Bork, A. (1981). *Learning with Computers*. Massachusetts: Digital Press. 481p.

Bork, A. (1987). *Learning with Personal Computers*. New York: Harper & Row Publishers. 238p.

Bork, A. (1988). New structures for technology-based courses. *Education and Computing*, 4. pp. 109-117.

Bork, A., Ibrahim B., Levrat, B., Milne, A. & Yoshii, R. (1992). The Irvine-Geneva course development system. In (ed.) Aiken, R. *Education and Society (Information Processing 92, Vol. 2).* pp. 253-261.

Bork, A. (1995a). Guest Editorial: Why has the computer failed in schools and universities? *Journal of Science Education and Technology*, 4(2). pp. 97-102.

Bork, A. (1995b). Rebuilding universities with highly interactive multimedia curricula. *Technical Report, Educational Technology Center, Information and Computer Science, University of California at Irvine, April 1995.* 48p.

Boutilier, C. (1994). Unifying default reasoning and belief revision in a modal framework. *Artificial Intelligence*, 68. pp. 33-85.

Boutilier, C. & Goldszmidt, M. (1993). Revision by conditional beliefs. *Proc. 11th National Conference on Artificial Intelligence.* pp. 649-654.

Broderick, W. R. & Lovatt, K. F. (1975). Acceptability of computer-managed instruction in the classroom - three years of experience. *Proceedings of the IFIP 2nd World Conf. on Computers in Education, Paris.* pp. 47-51.

Brophy, J. (1989, 1991, 1992). *Advances in Research on Teaching* (I, II, III). JAI Press Inc., Connecticut.

Brophy, J. & Alleman, J. (1992). Planning and managing learning activities: basic principles. In: (ed.) Brophy, J., *Advances in Research on Teaching*(3), pp. 1-45.

Brown, J. S. (1988). Towards a new epistemology for learning. In (eds.) Frasson, C. and Gauthier, G. *Intelligent tutoring systems: at the crossroads of artificial intelligence and education.* pp. 266-282. New Jersey: Ablex Publishing Corp.

Brown, J. S., Burton, R. R. & de Kleer, J. (1982). Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II, III. In Sleeman, D. and Brown, J. S. (Eds.), *Intelligent tutoring systems.* pp. 227-282. London: Academic Press.

Burton, R. R. (1982). Diagnosing bugs in a simple procedural skill. In Sleeman, D. and Brown, J. S. (Eds.), *Intelligent tutoring systems.* pp. 157-183. London: Academic Press.

Burton, R. R. & Brown, J. S. (1982). An investigation of computer coaching for informal learning activities. In Sleeman, D. and Brown, J. S. (Eds.), *Intelligent tutoring systems.* pp. 79-98. London: Academic Press.

Chan, T. & Baskin, A. B. (1988). Learning companion systems. In (eds.) Frasson, C. and Gauthier, G. *Intelligent tutoring systems: at the crossroads of artificial intelligence and education.* pp. 6-33. New Jersey: Ablex Publishing Corp.

Chen, Q. & Norcio, A. F. (1992). Modeling users with neural architectures. In *Proc. IEEE/INNS Int'l Joint Conf. on Neural Networks. Baltimore,* 1. pp. 547-552.

Clancey, W. J. (1987). *Knowledge-based tutoring: the GUIDON program.* Massachusetts: Massachusetts Institute of Technology.

Cohen, P. R. and Feigenbaum, E. A. (1982). (eds.) *The Handbook of Artificial Intelligence*, Vol. 3. California: William Kaufman Inc.

Colby, K. M. (1969). Experiments with a search algorithm for the data base of a human belief structure. *Proc. 1st Joint Conf. on Artificial Intelligence (IJCAI).* pp. 649-654.

Colby, K. M. (1975). *Artificial Paranoia: A Computer Simulation of Paranoid Processes.* New York: Pergamon Press. 113p.

Corbett, A. T.; Anderson, J. R. & Patterson, E. G. (1988). Student modeling and tutoring flexibility in the Lisp intelligent tutoring system. In (eds.) Frasson, C. and Gauthier, G. *Intelligent tutoring systems: at the crossroads of artificial intelligence and education.* pp. 83-106. New Jersey: Ablex Publishing Corp.

Criswell, E. L.; Boyer, M. A.; Canody, D. & Miller, L. (1990). Student-initiated interactions in a conventional CBI classroom. In *J. of computer-based instruction,* 17(2). pp. 61-65.

Dalal, M. (1988). Investigations into a theory of knowledge base revision: Preliminary report. *Proc. 7th National Conference on Artificial Intelligence (AAAI).* pp. 475-479.

de Kleer, J. (1986). An assumption-based TMS. *Artificial Intelligence*, 28. pp. 127-162.

Douglas, K. C.; Hosokawa, M. C. & Lawler, F. H. (1988). *A practical guide to clinical teaching in medicine.* New York: Springer Publishing Coy. 191p.

Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence*, 12. pp. 231-272.

Doyle, J. (1992). Reason maintenance and belief revision: Foundations vs. coherence theories. In: Gardenförs, P. (ed.). *Belief Revision.* Cambridge: Cambridge University Press. pp. 29-51.

Dunn, W. R. & Harden, R. M. (1971). Audio-visual self instruction in medical education. *Aspects of Educational Technology*, 5, pp. 90-115.

Ellis, A. B. (1974). *The Use and Misuse of Computers in Education.* New-York: McGraw-Hill Book Company. 226p.

Elton, L. R. B. (1971). Educational technology and the student explosion. *Aspects of Educational Technology*, 5, pp. 31-37.

Feifer, R. G. (1992). SHERLOCK: an intelligent tutoring system for teaching people how to learn. In (eds.) Kopec, D. and Thompson, R. B. *Artificial intelligence and intelligent tutoring systems: knowledge-based systems for teaching and learning.* pp. 111-127. London: Ellis Horwood.

Fennema, E., Carpenter, T.P. & Peterson, P.L. (1989). Learning with understanding: cognitively guided instruction. In: (ed.) Brophy, J., *Advances in Research on Teaching*(1), pp. 195-221.

Fiderio, J. (1988). A grand vision. *Byte*, 13(10). pp. 237-244.

Fikes, R. E. (1975). Deductive retrieval mechanisms for state description models. *Proc. 4th Joint Conf. on Artificial Intelligence (IJCAI).* pp. 99-106.

First, M.B.; Soffer, L.J & Miller, R.A. (1985). QUICK (QUIck Index to Caduceus Knowledge): using the Internist-1/Caduceus knowledge base as an electronic textbook of medicine. In *Computers and biomedical research* 18, pp. 137-165.

Foley, R.P. & Smilansky, J. (1980). *Teaching techniques: a handbook for health professionals.* McGraw-Hill Book Company: New York. 180p.

Fu, L. (1992). A neural network model for learning rule-based systems. In *Proc. IEEE/INNS Int'l Joint Conf. on Neural Networks. Baltimore,* 1. pp. 343-348.

Fujita, H.; Katafuchi, T.; Uehara, T. & Nishimura, T. (1992). Neural network approach for the computer-aided diagnosis of coronary artery diseases in nuclear medicine.

In *Proc. IEEE/INNS Int'l Joint Conf. on Neural Networks. Baltimore*, 3. pp. 215-220.

Frye, D., Littman, D. C. & Soloway, E. (1988). The next wave of problems in ITS: confronting the 'user issues' of interface design and system evaluation. In Psotka, J., Massey, L. D. and Mutter, S. A. (Eds.), *Intelligent tutoring systems: lessons learned.* pp. 451-478. Hillsdale, NJ: Lawrence Erlbaum Associates.

Gardenförs, P. (1984). Epistemic entrenchment and minimal changes of belief. *Australasian Journal of Philosophy*, 62(2). pp. 136-157.

Gardenförs, P. (1992). Belief revision: An introduction. In: Gardenförs, P. (ed.). *Belief Revision.* Cambridge: Cambridge University Press. pp. 1-28.

Genesereth, M. R. (1982). The role of plans in intelligent teaching systems. In Sleeman, D. and Brown, J. S. (Eds.), *Intelligent tutoring systems.* pp. 137-155. London: Academic Press.

Goodman, F. L. (1981). Computers and the future of literacy. *Proceedings of the AFIPS National Computer Conf., Chicago.* pp. 601-603.

Halasz, F.G. (1988). Reflections on NoteCards: seven issues for the next generation of hypermedia systems. In *Comm. ACM, 31(7),* July 1988. pp. 836-852.

Hand, D. J. (1985). *Artificial Intelligence and Psychiatry.* Cambridge: Cambridge University Press. 266p.

Hartman, A.; Diem, J. E. & Quagliana, M. (1992). The many faces of multimedia: how new technologies might change the nature of the academic endeavor. In (ed.) Barrett, E. *Sociomedia: multimedia, hypermedia, and the social construction of knowledge.* Massachusetts: pp. 175-191. MIT Press.

Hayes, P. J. (1975). A representation for robot plans. *Proc. 4th Joint Conf. on Artificial Intelligence (IJCAI).* pp. 181-188.

Heimler, C.; Cunningham, J. & Nevard, M. (1987). *Authoring Educational Software.* Santa Cruz: California: Mitchell Publishing, Inc. 271p.

Hills, P. J. (1971). Self teaching systems in university courses. *Aspects of Educational Technology*, 5, pp. 130-135.

Huang, X.; McCalla, G. I.; Greer, J. E. & Neufeld, E. (1991). Revising deductive knowledge and stereotypical knowledge in a student model. In (ed.) Kobsa, A. *User Modeling and User-Adapted Interaction,* 1(1). pp. 87-115. Dordrecht: Kluwer Academic Publishers.

Hume, G.D. (1992). A dynamic student model in a cardiovascular intelligent tutoring system. *Proceedings of the 5th Annual IEEE Symposium on Computer-Based Medical Systems.* pp. 370-377.

Jao, C. S. & Hier, D. B. (1993). Applying hypermedia and expert system technology to the neurological consultation. *Proceedings of the 6th Annual IEEE Symposium on Computer-Based Medical Systems*. pp. 118-123

Joshua, O. R. & Scuse, D. H. (1995). Tutoring strategies in the domain-independent adaptive tutoring system (DIATS). *Proceedings of the IASTED / ISMM International Conference on Intelligent Information Management Systems, Washington, D.C.* pp. 196-199.

Kaplan, H. I. & Sadock, B. J. (eds.,) (1985). *Comprehensive textbook of Psychiatry*. 4th Edition. Baltimore: Williams & Wilkins. 2129p.

Kass, R. & Finin, T. (1988). Modeling the user in natural language systems. In (eds.) Kobsa, A. and Wahlster, W. *Computational Linguistics*, 14(3). pp. 5-22. Massachusetts: MIT Press.

Kayser, D. & Coulon, D. (1979) CAI = Computer-Assisted Indoctrination? *Proceedings of the International Conference on Teleinformatics, Paris*. pp. 93-98.

Kochenburger, R. J. & Turcio, C. J. (1974). *Computers in Modern Society*. California: Hamilton Publishing Company. 266p.

Konologie, K. (1983). A deductive model of belief. *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI), Karlsruhe, FRG.* pp. 377-381.

Kopec, D.; Brody, M.; Shi, A. C. & Wood, C. (1992). Towards an intelligent tutoring system with application to sexually transmitted diseases. In (eds.) Kopec, D. and Thompson, R. B. *Artificial intelligence and intelligent tutoring systems: knowledge-based systems for teaching and learning*. pp. 129-151. London: Ellis Horwood.

Kurland, L.C.; Granville, R.A. & MacLaughlin, D.M. (1992). Design, development and implementation of an intelligent tutoring system for training radar mechanics to troubleshoot. In (eds.) Farr, M.J. and Psotka, J. *Intelligent instruction by computer: theory and practice*. pp. 205-237. Washington: Taylor & Francis New York, Inc.

Lajoie, S. & Lesgold, A. (1992). Apprenticeship training in the workplace: computer-coached practice environment as a new form of apprenticeship. In (eds.) Farr, M. J. and Psotka, J. *Intelligent instruction by computer: theory and practice*. pp. 15-36. Washington: Taylor & Francis New York, Inc.

Lampert, M. (1989). Choosing and using mathematical tools in classroom discourse. In: (ed.) Brophy, J., *Advances in Research on Teaching*(1), pp. 223-264.

Landauer, T. K. & Freedman, J. L. (1968). Information retrieval from long-term memory: Category size and recognition time. *Journal of Verbal Learning and Verbal Behavior*, 7. pp. 291-295.

Lanza, A. & Roselli, T. (1991). Effects of the hypertextual approach versus the structured approach on student's achievement. In *J. of computer-based instruction*, 18(2). pp. 48-50.

Larsen, R. E. (1992). Relationship of learning style to the effectiveness and acceptance of interactive video instruction. In *J. of computer-based instruction,* 19(1). pp. 17-21.

Laver, F. J. M. (1976). *An Introduction to the Uses of Computers.* London: Cambridge University Press. 376p.

Lawler, R. W. & Yazdani, M. (1987). (eds.) *Artificial Intelligence and Education: learning environments and tutoring systems.* New-Jersey: Ablex Publishing Company.

Leitch, R. R.; Ponnapalli, P. & Slater, A. F.(1992). The representation of domain knowledge in intelligent tutoring systems. In *First IEE Int'l Conf. on intelligent systems. Edinburgh.* pp. 269-275. London: IEE

Levin, S. R. (1991). The effects of interactive video enhanced earthquake lessons on achievement of seventh grade earth science students. In *Journal of computer-based instruction,* 18(4). pp. 125-129.

Liebmann, M., Mackie, A., & Glover, C. (1971). Writing programs for unstreamed classes. *Aspects of Educational Technology*, 5, pp. 144-153.

Lincoln, M. A., & McAllister, L. L. (1993). Peer learning in clinical education. In (ed.) Harden, R. M. *Medical Teacher,* 15(1). pp. 17-25. Journals Oxford Ltd.

Los Arcos, J. L.; Angulo, I.; Sanchez, V.; Sabugo, M. J. & Burguera, E. (1992). Power plant domain representation for operator intelligent training systems. In *First IEE Int'l Conf. on intelligent systems. Edinburgh.* pp. 250-256. London: IEE.

Maida, A. & Shapiro, S. (1982). Intensional concepts in propositional semantic networks. In *Cognitive Science*, 6(4), pp. 291-330.

Makinson, D. (1985). How to give it up: a survey of some formal aspects of the logic of theory change. *Synthèse*, 62. pp. 347-363.

Maren, A. J. (1991). Neural networks for enhanced human-computer interactions. In *IEEE Control systems,* 11(5). pp. 34-36.

Markle, S. M. (1969). *Good Frames and Bad: a grammar of frame-writing.* 2nd Edition. New-York: John Wiley and Sons, Inc. 308p.

Martins, J. (1992). Belief revision. In (ed.) Shapiro, S. C. *Encyclopedia of Artificial Intelligence*, 2. (2nd. edition). New York: John Wiley and Sons. pp. 111-116.

Martins, J. P. & Shapiro, S. C. (1988). A model of belief revision. *Artificial Intelligence*, 35. pp. 25-79.

McCalla, G. I.; Greer, J. E. & SCENT Research Team (1988). SCENT-3: an architecture for intelligent advising in problem-solving domains. In (eds.) Frasson, C. and Gauthier, G. *Intelligent tutoring systems: at the crossroads of artificial intelligence and education.* pp. 140-161. New Jersey: Ablex Publishing Corp.

McCarthy, J. & Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In (eds.) Meltzer, B. and Michie, D. *Machine Intelligence*, 4, pp. 463-502. Edinburgh: Edinburgh University Press.

Miller, M. L. & Lucado, S. R. (1992). Integrating intelligent tutoring, computer-based training, and interactive video in a prototype maintenance trainer. In (eds.) Farr, M. J. and Psotka, J. *Intelligent instruction by computer: theory and practice*. pp. 127-150. Washington: Taylor & Francis New York, Inc.

Moreno, H. R. & Plant, R. T. (1993). A prototype decision support system for differential diagnosis of psychotic, mood and organic disorders. In (ed..) Beck, J. R. *Medical Decision Making*, 13(1). pp. 43-48. Philadelphia: Hanley & Belfus.

Nwana, H. S. (1990a). Intelligent tutoring systems: an overview. In *Artificial intelligence review*, 4. pp. 251-277.

Nwana, H. S. (1990b). The anatomy of FITS: a mathematics tutor. In *Intelligent tutoring media*, 1(2). pp. 83-95.

Nwana, H. S. (1991a). The evaluation of an intelligent tutoring system. In *Intelligent tutoring media*, 1(3). pp. 1-24.

Nwana, H. S. (1991b). User modeling and user-adapted interaction in an intelligent tutoring system. In (ed.) Kobsa, A. *User modeling and user-adapted interaction*, 1(1). pp. 1-32. Dordrecht: Kluwer Academic Publishers.

Nwana, H. S. (1991c). FITS: a fraction intelligent tutoring system. In *Proc. 9th National Conf. on Artificial Intelligence*, 1. pp. 49-54. Menlo Park: MIT Press.

Nwana, H. S. (1993). An approach to developing intelligent tutors in mathematics. In *Computers educ.*, 20(1). pp. 27-43.

Nwana, H. & Coxhead, P. (1988). Towards an intelligent tutor for a complex mathematical domain. In *Expert systems*, 5(4). pp. 290-300.

Nwana, H. S. & Coxhead, P. (1989). Fraction bugs: explanations, bug theories, and implications on intelligent tutoring systems. In *Cognitive systems*, 2(3). pp. 275-289.

Ohlsson, S. (1987). Some principles of intelligent tutoring. In Lawler, R. W. and Yazdani, M. (Eds.), Learning environments and tutoring systems. *Artificial intelligence and education*, 1, 203-237. New Jersey: Albex.

Papadourakis, G. M.; Gaga, E; Vareltzis, G & Bebis, G. (1992). Use of artificial neural networks for clinical decision-making (maldescensus testis). In *Proc. IEEE/INNS Int'l Joint Conf. on Neural Networks. Baltimore*, 3. pp. 159-164.

Paris, C. L. (1988). Tailoring object descriptions to a user's level of expertise. In (eds.) Kobsa, A. and Wahlster, W. *Computational Linguistics*, 14(3). pp. 64-78. Massachusetts: MIT Press.

Park, O. (1991). Hypermedia: functional features and research issues. In *Educational technology*, 31(8). pp. 24-31.

Parkes, A. P. & Self, J. A. (1988). Towards "interactive video": a video-based intelligent tutoring environment. In (eds.) Frasson, C. and Gauthier, G. *Intelligent tutoring systems: at the crossroads of artificial intelligence and education.* pp. 56-82 New Jersey: Ablex Publishing Corp.

Peterson, P.L., Fennema, E. & Carpenter, T. P. (1991). Teachers' knowledge of students' mathematics problem-solving knowledge. In: (ed.) Brophy, J., *Advances in Research on Teaching*(2), pp. 49-86.

Price, R. V. (1991). *Computer-aided instruction: a guide for authors.* Pacific Grove, California: Brooks/Cole Publishing Coy. 386p.

Rada, R. (1991). *Hypertext: from text to expertext.* London: McGraw-Hill Book Company. 237p.

Rapaport, W. J. (1986). Logical foundations for belief representation. In (ed.) Waltz, D. L. *Cognitive Science,* 10. pp. 355-366. New Jersey: Ablex Publishing Corp.

Rapaport, W. J. (1992). Belief representation systems. In (ed.) Shapiro, S. C. *Encyclopedia of Artificial Intelligence*, 2. (2nd. edition). New York: John Wiley and Sons. pp. 98-111.

Redish, E. F.; Wilson, J. M. & McDaniel, C. (1992). The CUPLE project: a hyper- and multimedia approach to restructuring Physics education. In (ed.) Barrett, E. *Sociomedia: multimedia, hypermedia, and the social construction of knowledge.* Massachusetts: pp. 219-255. MIT Press.

Reggia, J. A., Nau, D. S. & Wang, P. Y. (1983). Diagnostic expert systems based on a set covering model. *International Journal of Man-Machine Studies*, 19, pp. 437-460.

Rich, E. (1979). User modeling via stereotypes. In (eds.) Schank, R.; Collins, A. and Charniak, E. *Cognitive Science,* 3. pp. 329-354. New Jersey: Ablex Publishing Corp.

Rich, E. & Knight, K. (1991). *Artificial Intelligence*. 2nd Edition. New-York: McGraw-Hill, Inc. 621p.

Rickel, J. W. (1989). Intelligent computer-aided instruction: a survey organized around system components. In *IEEE Transactions on systems, man, and cybernetics,* 19(1). pp. 40-57.

Rieber, L. P.; Boyce, M. J & Assad, C. (1990). The effects o computer animation on adult learning and retrieval tasks. In *J. of computer-based instruction,* 17(2). pp. 46-52.

Rieber, L. P. & Kini, A. S. (1991). Theoretical foundations of instructional applications of computer-generated animated visuals. In *J. of computer-based instruction,* 18(3). pp. 83-88.

Sanders, D. H. (1979). *Computers in Business: An Introduction.* 4th Edition. Tokyo: McGraw-Hill Book Company. 706p.

Schustack, M. W. & Anderson, J. R. (1979). Effects of analogy to prior knowledge on memory for new information. *Journal of Verbal Learning and Verbal Behavior*, 18. pp. 565-583.

Self, J. A. (1988). Bypassing the intractable problem of student modeling. In (eds.) Frasson, C. and Gauthier, G. *Intelligent tutoring systems: at the crossroads of artificial intelligence and education.* pp. 107-123. New Jersey: Ablex Publishing Corp.

Shim, L.; Evens, M.W.; Michael, J.A. & Rovick, A.A (1991). Effective cognitive modeling in an intelligent tutoring system for cardiovascular physiology. *Proceedings of the 4th Annual IEEE Symposium on Computer-Based Medical Systems.* pp. 338-345

Shneiderman, B. (1992). *Designing the user interface: strategies for effective human-computer interaction.* 2nd Edition. Massachusetts: Addison-Wesley.

Sime, J-A. & Leitch, R. (1992). Multiple models in intelligent tutoring, In *First IEE Int'l Conf. on intelligent systems. Edinburgh.* pp. 263-268. London: IEE

Sleeman, D. (1987). PIXIE: a shell for developing intelligent tutoring systems. In Lawler, R. W. and Yazdani, M. (Eds.), Learning environments and tutoring systems. *Artificial intelligence and education*, 1, 239-263. New Jersey: Albex.

Sleeman, D. & Brown, J. S. (1982). (eds.) *Intelligent Tutoring Systems.* London: Academic Press.

Smith, J.B. & Weiss, S.F. (1988). Hypertext. In *Comm. ACM, 31(7),* July 1988. pp. 816-819.

Soloway, E. (1992). An emerging technology gears up. In (eds.) Farr, M. J. and Psotka, J. *Intelligent instruction by computer: theory and practice.* pp. 265-268. Washington: Taylor & Francis New York, Inc.

Spensley, F.; Elsom-Cook, M.; Byerley, P.; Brooks, P.; Federici, M. & Scaroni, C. (1988). Using multiple teaching strategies in an ITS. In (eds.) Frasson, C. and Gauthier, G. *Intelligent tutoring systems: at the crossroads of artificial intelligence and education.* pp. 188-205. New Jersey: Ablex Publishing Corp.

Stallman, R. M. & Sussman, G. J. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9. pp. 135-196.

Stevens, A.; Collins, A. & Goldin, S. E. (1982). Misconceptions in students' understanding. In Sleeman, D. and Brown, J. S. (Eds.), *Intelligent tutoring systems.* pp. 13-24. London: Academic Press.

Tennyson, R. D. & Park, O. C. (1987). Artificial intelligence and computer-based learning. In Gagne, R. M. (Ed.), *Instructional technology: foundations.* pp. 319-342. Hillsdale, NJ: Lawrence Erlbaum Associates.

The American Psychiatric Association (1987). *Diagnostic and statistical manual of mental disorders (DSM-III-R).* 3rd Edition (Revised). Washington: American Psychiatric Association. 567p.

Underwood, B. J. & Ekstrand, B. R. (1968). Linguistic associations and retention. *Journal of Verbal Learning and Verbal Behavior*, 7. pp. 162-171.

van den Berg, S. & Watt, J. H.(1991). Effects of educational setting on student responses to structured hypertext. In *J. of computer-based instruction,* 18(4). pp. 118-124.

Venezky, R. & Osin, L. (1991). *The Intelligent Design of Computer-Assisted Instruction.* New-York: Longman Publishing Group. 336p.

Wang, S. & el Ayeb, B. (1992). Diagnosis: hypothetical reasoning with a competition-based neural architecture. In *Proc. IEEE/INNS Int'l Joint Conf. on Neural Networks. Baltimore,* 1. pp. 7-12.

Waterman, D. A. (1986). *A guide to expert systems.* Massachusetts: Addison-Wesley Publishing Company. 419p.

Winnie, P. P. & Kramer, L. L. (1988). Representing knowledge about teaching: DOCENT - an AI planning system for teaching and learning. In (eds.) Frasson, C. and Gauthier, G. *Intelligent tutoring systems: at the crossroads of artificial intelligence and education.* pp. 162-187. New Jersey: Ablex Publishing Corp.

Woo, C W.; Evens, M.; Michael, J. & Rovick, A. (1991). Dynamic instructional planning for an intelligent physiology tutoring system. *Proceedings of the 4th Annual IEEE Symposium on Computer-Based Medical Systems.* pp. 226-233

Woolf, B. (1992). Hypermedia in education and training. In (eds.) Kopec, D. and Thompson, R. B. *Artificial intelligence and intelligent tutoring systems: knowledge-based systems for teaching and learning.* pp. 97-109. London: Ellis Horwood.

Yazdani, M. (1987). Intelligent tutoring systems: an overview. In Lawler, R. W. and Yazdani, M. (Eds.), Learning environments and tutoring systems. *Artificial intelligence and education*, 1, pp. 183-201. Hillsdale, NJ: Lawrence Erlbaum Associates.