

December 1988

Report No. STAN-CS-88-1242

Also Numbered KSL-88-14

Thesis

1

AD-A225 044

DTIC FILE COPY

# Apprenticeship Learning Techniques for Knowledge Based Systems

DTIC  
ELECTE  
JUL 30 1990  
S D *CS* D

by

David Chester Wilkins

Department of Computer Science

Stanford University

Stanford, California 94305

UNION STATEMENT A  
Approved for public release  
Distribution Unlimited



**APPRENTICESHIP LEARNING TECHNIQUES  
FOR  
KNOWLEDGE BASED SYSTEMS**

by

David Chester Wilkins

A dissertation submitted in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy  
(Computer Science and Engineering)  
in The University of Michigan  
1987

Doctoral Committee:

Professor Bruce G. Buchanan, Co-Chair  
Stanford University  
Professor John H. Holland, Co-Chair  
Assistant Professor Paul D. Scott, Co-Chair  
Professor Emeritus Arthur W. Burks  
Assistant Professor John E. Laird  
Professor Robert K. Lindsay

© David Chester Wilkins 1987  
All Rights Reserved

<b>BIBLIOGRAPHIC DATA SHEET</b>	1. Report No.	2.	3. Recipient's Accession No.
4. Title and Subtitle Apprenticeship Learning Techniques for Knowledge Based Systems		5. Report Date December, 1988	
7. Author(s) David C. Wilkins		6.	
9. Performing Organization Name and Address Department of Computer Science Stanford University Stanford, CA 94305		8. Performing Organization Rep. No.	
		10. Project/Task/Work Unit No.	
		11. Contract/Grant No.	
12. Sponsoring Organization Name and Address National Science Foundation Washington, D.C. 20550 Office of Naval Research Arlington, VA 22202		13. Type of Report & Period Covered	
		14.	
15. Supplementary Notes			
16. Abstracts <p>This thesis describes <i>apprenticeship learning techniques</i> for automation of the transfer of expertise. The major accomplishment in this thesis is showing how an explicit representation of the strategy knowledge to solve a general problem class, such as diagnosis, can provide a basis for learning the knowledge that is specific to a particular domain, such as medicine. The Odysseus explanation-based learning program constructs explanations of problem-solving actions in the domain of medical diagnosis. If no explanation is found, the incomplete domain theory (i.e., the medical knowledge base) is extended via the use of underlying domain theories and empirical methods so as to allow construction of an explanation. The Odysseus learning program provides the first demonstration of using the same technique to transfer of expertise <i>to</i> and <i>from</i> an expert system knowledge base. When watching an expert, it improves a knowledge base for the pre-existing Heracles expert system shell. When watching a student apprentice, it models the student against the knowledge base and thereby identifies bugs and gaps in the student's fledgling expertise.</p> <p>Another major focus of this thesis is <i>limitations of apprenticeship learning</i>. It is shown that extant techniques for reasoning under uncertainty for expert systems lead to a <i>sociopathic knowledge base</i>, wherein a subset of the knowledge base can give better performance than the original knowledge base; incremental learning techniques are inappropriate when a knowledge base is sociopathic. Also, the <i>synthetic agent method</i> is presented; it provides a means of determining a performance upper bound for apprenticeship learning systems.</p>			
17. Key Words <p>Artificial Intelligence, Knowledge Based Systems, Expert Systems, Machine Learning, Apprenticeship Learning, Explanation Based Learning, Incomplete Domain Theory, Sociopathic Knowledge Base, Medical Diagnosis, Intelligent Tutoring, Cognitive Modeling.</p>			
17c. COSATI Field/Group			
18. Availability Statement unlimited		19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
		20. Security Class (This Page) UNCLASSIFIED	22. Price

# ABSTRACT

## APPRENTICESHIP LEARNING TECHNIQUES FOR KNOWLEDGE BASED SYSTEMS

by

David Chester Wilkins

December, 1987

The significance of machine learning for the future use of computers is very great. Autonomous computer systems of the future will need far more knowledge than humans can explicitly transfer; this requires that computers learn independently. An important research goal for machine learning is to identify techniques that will allow intelligent knowledge-based systems to learn automatically the large amounts of domain-specific knowledge that are necessary for achieving expert-level problem-solving performance.

This thesis describes *apprenticeship learning techniques* for automation of the transfer of expertise. The major accomplishment in this thesis is showing how an explicit representation of the strategy knowledge to solve a general problem class, such as diagnosis, can provide a basis for learning the knowledge that is specific to a particular domain, such as medicine. The Odysseus explanation-based learning program is presented, which constructs explanations of human problem-solving actions in the domain of medical diagnosis. If no explanation can be constructed, Odysseus extends the incomplete domain theory (i.e., the medical knowledge base) via the use of underlying domain theories and empirical methods so as to allow construction of an explanation. Odysseus provides the first demonstration of using the same technique to transfer of expertise *to* and *from* an expert system knowledge base. When watching an expert, it improves a knowledge base for the pre-existing Heracles expert system shell. When watching a student apprentice, it models the student against the knowledge base and thereby identifies bugs and gaps in the student's fledgling expertise.

Another major focus of this thesis is *limitations of apprenticeship learning*. It is shown that extant techniques for reasoning under uncertainty for expert systems lead to a *sociopathic knowledge base*, wherein a subset of the knowledge base can give better performance than the original knowledge base; incremental learning techniques are inappropriate when a knowledge base is sociopathic. Also, the *syntheti agent method* is presented; it provides a means of determining a performance upper bound for apprenticeship learning systems.



A-1

to Marianne

## ACKNOWLEDGMENTS

My thesis has been heavily influenced by all my committee members. I am grateful to Bruce Buchanan for instilling an understanding of how one does research and what constitutes good research. The freedom Bruce provided me to pursue my own ideas has been priceless to my development. Bill Clancey has played an equally significant role in this work. My ODYSSEUS learning program builds upon Bill's NEOMYCIN expert system, a program that mirrors Bill's deep understanding of the important issues and directions in expert systems research. Paul Scott's knack for totally dissecting my techniques and arguments has always astonished me, and has greatly improved this thesis. The Logic of Computers group of Art Burks and John Holland was an inspiration to me many years before I started graduate school. Art and John are my models for good scientists, and doing work that represents achievement by their standards is my lifetime goal. Bob Lindsay's emphasis on taking a scientific approach and formulating testable hypotheses led me to abandon several earlier thesis proposals, and I am very grateful for the perspective he provided. John Laird kindly agreed to join my committee on short notice and provided valuable comments on my thesis draft.

During the evolution of this thesis, I received sage advice periodically from my models of lucid thinking: Tom Dietterich, Tom Mitchell, Paul Rosenbloom, and Kurt VanLehn. Their observations always deepened my understanding and led to improved presentation of my methods and results. I am also indebted to Pat Langley, John McDer-

mott, Ryszard Michalski, Roy Rada and Derek Sleeman for thesis-related discussions that were of significant help. My early thesis formulation was greatly aided by discussions with Saul Amarel, Avron Barr and Jim Bennett. Avron once considered doing a dissertation on learning within NEOMYCIN, and his advice on the challenges of such an endeavor was helpful.

I appreciate the comradeship and useful advice of my GRAIL Learning Group cohorts, especially Tom Dietterich, Andy Golding, Benjamin Groszof, Haym Hirsh, Peter Karp, Paul Rosenbloom, Devika Subramanian, and Stuart Russell. They made exploring the terrain of machine learning a lot of fun. In addition, Devika, Haym, and Marianne Winslett critiqued many drafts of many papers that have become part of this thesis; their suggestions significantly improved the presentation and content.

Many members of the GUIDON Tutoring Group contributed code that was directly or indirectly used by my ODYSSEUS learning program, and I would like to thank Steve Barnhouse, Conrad Bock, Diane Hasling, David Leserman, Arif Merchant, Steven Oliphant, Mark Richer, and Naomi Rodolitz.

For discussions on complexity analysis and uncertainty reasoning I am grateful to Ramsey Haddad, David Heckerman, Eric Horovitz, Curt Langlotz, Peter Rathmann, and Marianne Winslett. Jim Koehler and David Shapiro provided helpful statistical advice.

Many physicians have played an instrumental role in this research. Curt Kapsner and John Sotos taught me most of what I know about medicine and provided frequent medical advice. John and Edward Herskovits helped me create a case library for the NEOMYCIN domain from medical records at the Stanford University Hospital, thereby allowing evaluation of my ODYSSEUS learning program. Ted Shortliffe made many helpful suggestions and constructed a disease hierarchy for the MYCIN domain that was used in experiments to measure improvement caused by reduction of sociopathic interactions.

I am grateful to Mark Frisse for suggesting the term 'sociopathic'. John Sotos, Larry Fagan, Mark Musen, Randy Miller, and Roy Rada provided problem-solving protocols that were used in validation experiments. In addition, a number of medical students provided problem-solving protocols, and I would like to especially thank Russ Altman.

At Xerox PARC, the Tuesday paper-lunch group provided a sharp and provocative counterpoint to the conception of knowledge based systems that I absorbed at Stanford's Knowledge Systems Lab. I would like to thank John Seely Brown, Jim Greeno, Kurt VanLehn, and Jeff Shrager for the interest they took in my work. Most of the programming of ODYSSEUS was done at PARC's Intelligent Systems Lab.

Doubtless the writing of this thesis would have cost me my sanity if it were not for the members of Saint Ann's Choir. I would especially like to thank Bill Mahrt, Eunice Schroeder, and Ellen Pint. They took care of my spiritual health, while the SCRA Sunday morning tennis group, with Bruce Buchanan, Bob Engelmores, and Curt Langlotz, preserved my physical health. My parents, brother, and sisters have seen much less of me during the last few years of writing this thesis. I would like to thank Joseph, Clare, Patricia, Timothy, and Mary Beth, who are all very special to me, for their understanding and encouragement.

Marianne Winslett provided constant emotional and intellectual support in the writing of this dissertation, and helped weave frequent visits to the Pacific Ocean, the Sierras, and San Francisco into its tapestry. It is to her that this thesis is dedicated.

While the techniques developed in my thesis originated with me, the research directly builds on past work at Stanford's Knowledge Systems Lab (formerly, the Heuristic Programming Project), and thereby owes an enormous intellectual debt to the work of Ed Feigenbaum, Bruce Buchanan, Ted Shortliffe, Randy Davis, Bill Clancey, and Mike Genesereth. Appendix D traces the genealogy of the ODYSSEUS learning program that is

the centerpiece of this thesis.

The computer time provided by SUMEX-AIM is appreciated, as well as the efforts of Tom Rindfleisch, Christopher Schmidt, and Bill Yeager to provide a superb computing environment. To obtain the full aesthetic benefits of T<sub>E</sub>X required frequent visits to the mind of its creator, Donald Knuth, and this introduced into the writing a way of thinking that is quite alien to artificial intelligence.

This work was principally supported by the National Science Foundation under grant MCS-83-12148 and the Office of Naval Research under contracts N00014-79C-0302 and N00014-88K-0124. I am appreciative of Susan Chipman's interest in the cognitive modeling aspects of my work. Support was also received from the Advanced Research Projects Agency under contract N00039-83-C-0136, the National Institute of Health under grant NIH RR-00785-11, and the National Aeronautics and Space Administration under grant NAG-5-261.

## TABLE OF CONTENTS

DEDICATION . . . . .	ii
ACKNOWLEDGMENTS . . . . .	iii
LIST OF FIGURES . . . . .	x
LIST OF TABLES . . . . .	xi
LIST OF APPENDICES . . . . .	xii
CHAPTER	
1 INTRODUCTION . . . . .	1
1.1 The Problem and Results . . . . .	2
1.2 Odysseus Apprenticeship Learning Program . . . . .	5
1.2.1 The Performance Model of Learning . . . . .	5
1.2.2 Odysseus' Learning Methodology . . . . .	8
1.3 Example of Learning by Watching . . . . .	11
1.3.1 Stage 1: Detecting Deficiencies in a Knowledge Base . . . . .	11
1.3.2 Stage 2: Suggesting Repairs to a Knowledge Base . . . . .	12
1.3.3 Stage 3: Evaluating Repairs to a Knowledge Base . . . . .	13
1.4 Reader's Guide . . . . .	14
2 BACKGROUND . . . . .	15
2.1 Problem-Solving Architecture of Heracles . . . . .	15
2.1.1 Object-Level Representation . . . . .	16
2.1.2 Syntax of Object-Level Knowledge . . . . .	16
2.1.3 Specification of Object-Level Knowledge . . . . .	18
2.1.4 Semantics of Object-Level Knowledge . . . . .	20
2.1.5 Meta-Level Representation . . . . .	20
2.2 Related Research . . . . .	21
2.2.1 Knowledge Acquisition . . . . .	21
2.2.2 Machine Learning . . . . .	24
2.2.3 Student Modeling . . . . .	26
2.2.4 Neomycin-based Modeling Programs . . . . .	28

2.2.5	Plan Recognition and Language Understanding . . . . .	30
2.2.6	Automatic Programming . . . . .	31
<b>3</b>	<b>DETECTING DEFICIENCIES IN A KNOWLEDGE BASE . . . . .</b>	<b>32</b>
3.1	Methods of Detecting Deficiencies . . . . .	33
3.1.1	Problem Solution as External Standard . . . . .	33
3.1.2	Problem Solution Steps as External Standard . . . . .	34
3.1.3	Environmental Feedback as External Standard . . . . .	35
3.1.4	Oracle as External Standard . . . . .	36
3.1.5	Efficiency as Internal Standard . . . . .	36
3.1.6	Correctness as Internal Standard . . . . .	37
3.2	Detecting Deficiencies When Learning by Watching . . . . .	37
3.2.1	Finding Explanations by Reverse Interpretation . . . . .	38
3.2.2	The Strategy Space of Heracles . . . . .	40
3.3	Detecting Deficiencies When Learning from Experience . . . . .	42
<b>4</b>	<b>SUGGESTING REPAIRS TO A KNOWLEDGE BASE . . . . .</b>	<b>45</b>
4.1	Techniques for Suggesting Repairs . . . . .	45
4.1.1	Mutation and Crossover . . . . .	46
4.1.2	Weight Adjustment . . . . .	47
4.1.3	Rule Generalization . . . . .	47
4.1.4	Plan Generalization . . . . .	47
4.1.5	Plan Completion . . . . .	47
4.1.6	Analogy . . . . .	48
4.1.7	Oracle . . . . .	48
4.2	Suggesting Repairs When Learning by Watching . . . . .	48
4.3	Suggesting Repairs When Learning From Experience . . . . .	49
4.4	Example of Suggesting Repairs . . . . .	50
<b>5</b>	<b>EVALUATING REPAIRS TO A KNOWLEDGE BASE . . . . .</b>	<b>55</b>
5.1	Spectrum of Techniques for Evaluating Repairs . . . . .	56
5.2	The Confirmation Theory . . . . .	56
5.2.1	Knowledge- vs. Performance-oriented Validation . . . . .	57
5.2.2	Examples of Using a Confirmation Theory . . . . .	59
5.2.3	Confirmation Theory for Factual Knowledge . . . . .	60
5.2.4	Confirmation Theory for Rule Knowledge . . . . .	63
5.3	The Underlying Domain Theory . . . . .	66
5.3.1	Odysseus' Induction System . . . . .	66
5.3.2	Related Research . . . . .	67
5.4	Example of Evaluating Repairs . . . . .	67

<b>6</b>	<b>INHERENT LIMITS OF LEARNING</b>	<b>70</b>
6.1	Sociopathic Knowledge Bases	71
6.1.1	Reasoning Under Uncertainty	72
6.1.2	Inexact Reasoning and Rule Interactions	74
6.2	Debugging Sociopathic Knowledge Bases	77
6.2.1	Types of Rule Interactions	77
6.2.2	Traditional Methods of Debugging a Rule Set	78
6.3	Minimizing Sociopathic Interactions	79
6.3.1	Bipartite Graph Minimization Formulation	80
6.3.2	Sociopathic Reduction Algorithm	82
6.3.3	Example of Sociopathic Reduction	85
6.3.4	Experimental Results	87
6.4	Summary	88
<b>7</b>	<b>UPPER LIMITS OF LEARNING</b>	<b>89</b>
7.1	Evaluation and the Synthetic Agent Method	91
7.2	Synthetic Agent Method of Evaluation	92
7.2.1	The Synthetic Agent Method	95
7.2.2	Discussion of Synthetic Agent Method	97
7.2.3	Categories of Errors	98
7.3	Summary	99
<b>8</b>	<b>LOWER LIMITS OF LEARNING</b>	<b>101</b>
8.1	Knowledge Acquisition Results	103
8.2	Student Modeling Results	106
8.3	Comparing Apprentice Scenarios	108
<b>9</b>	<b>CONCLUSIONS</b>	<b>109</b>
9.1	Contributions	109
9.1.1	Techniques for Apprenticeship Learning	109
9.1.2	Fundamental Limits of Learning	111
9.1.3	Symmetry of Learning and Teaching	112
9.2	Main Limitations of Approach	112
9.2.1	Underconstrained Interpretations of Actions	113
9.3	Further Work	116
	<b>APPENDICES</b>	<b>118</b>
	<b>BIBLIOGRAPHY</b>	<b>127</b>
	<b>INDEX</b>	<b>136</b>

## LIST OF FIGURES

### Figure

1.1	The learning by watching apprenticeship scenario . . . . .	3
1.2	The three principal apprenticeship scenarios . . . . .	4
1.3	The performance model of learning . . . . .	6
1.4	Odysseus' method for learning by watching . . . . .	10
1.5	An example of what the Odysseus apprentice learner sees . . . . .	12
2.1	Problem-solving architecture of Heracles . . . . .	17
3.1	Odysseus' method of detecting discrepancies . . . . .	38
3.2	Degrees of strategy differences . . . . .	41
3.3	The learning from experience apprenticeship scenario . . . . .	42
3.4	Odysseus' method for learning from experience . . . . .	43
5.1	A use-independent knowledge base . . . . .	57
5.2	Types of knowledge that improve performance . . . . .	58
6.1	Bipartite graph formulation of sociopathic interactions . . . . .	83
6.2	Example of bipartite graph formulation for one hypothesis . . . . .	86
7.1	The synthetic agent method for learning by watching . . . . .	93
7.2	Different categories of knowledge . . . . .	94
D.1	Genealogy of the Odysseus apprenticeship learning program . . . . .	124

## LIST OF TABLES

### Table

3.1	Performance standards for detecting deficiencies used by learning programs .	33
4.1	Methods of suggesting repairs used by learning programs . . . . .	46
5.1	Methods of evaluating repairs used by learning programs . . . . .	55
8.1	Performance of Neomycin before learning . . . . .	104
8.2	Performance of Neomycin after learning . . . . .	105
8.3	Performance improvement from learning by watching . . . . .	106

## LIST OF APPENDICES

### Appendix

A	ODYSSEUS PROGRAM . . . . .	119
B	HERACLES PROGRAM . . . . .	121
C	CALCULATING $\Phi$ . . . . .	123
D	GLOSSARY . . . . .	124

# CHAPTER 1

## INTRODUCTION

The significance of machine learning for the future use of computers is very great. Autonomous computer systems of the future will need far more knowledge than humans can explicitly transfer; this requires that computers learn independently. An important research goal for machine learning is to identify techniques that will allow intelligent knowledge-based systems to automatically learn the large amount of domain-specific knowledge that is necessary to achieve expert-level problem-solving performance.

Apprenticeship is the most effective means for human problem solvers to learn or teach domain-specific problem-solving knowledge in knowledge-intensive domains. This observation provides motivation to give apprenticeship learning abilities to knowledge-based expert systems. The paradigmatic example of an apprenticeship period is medical training. In medicine, high-level performance is obtained by spending two or three years acquiring 'first-principles' textbook knowledge (beginning-game knowledge acquisition) and then spending five to eight years in an apprenticeship role, called a clerkship, an internship, and a residency (end-game knowledge acquisition). An apprenticeship period succeeds in creating human experts in some knowledge-intensive professions, and hence it makes sense to investigate its efficacy in creating artificial experts.

## 1.1 The Problem and Results

This dissertation describes *apprenticeship learning techniques* for automation of the transfer of expertise to and from an expert system knowledge base, and describes fundamental limitations of an apprenticeship learning approach. By definition, apprentice learning systems improve a performance program in the course of *normal problem solving* and derive their power from the use of an *underlying domain theory* (Mitchell et al., 1985). Our experience in implementing an apprenticeship learning program in the domain of diagnosing neurological problems has yielded four general results. First, a demonstration is provided of how an explicit representation of meta-level strategy knowledge for a general problem class, such as diagnosis, can provide a basis for learning the object-level knowledge that is specific to a particular domain, such as medical knowledge. Second, we demonstrate the need for and use of a *confirmation theory* to mediate between the object-level knowledge to be learned and the underlying domain theory. A confirmation theory provides a decision procedure for judging potential repairs to the knowledge base, and is required when deleterious interactions can occur between facts or rules of a 'use-independent' knowledge base during problem solving. Third, it is shown that there are *inherent limits to learning* for apprenticeship techniques, in particular, and incremental learning techniques, in general, for a wide class of knowledge bases. In addition to inherent learning limits, we present results concerning the calculation of an upper and lower performance bound on an apprentice learning system. Lastly, we provide the first demonstration of using the same technique to transfer expertise both *to* and *from* the knowledge base of an expert system. When watching an expert, the ODYSSEUS learning program improves an expert system. When watching a student, the ODYSSEUS learning program improves the student's fledgling expertise.

There are three principal apprenticeship learning scenarios used by human problem

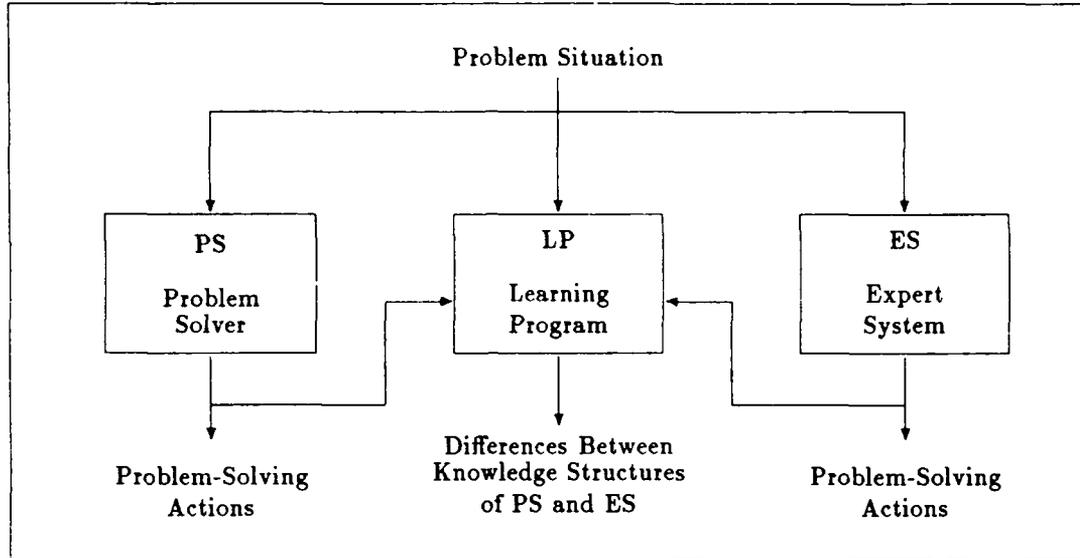


Figure 1.1: The *learning by watching* apprenticeship scenario. The learning program models an expert system against an expert or student problem solver in order to detect discrepancies between the knowledge of the human and expert system. The learning program then uses the meta-level strategy knowledge of the expert system to suggest object-level differences in the knowledge structures. These suggestions are tested using an underlying domain theory.

solvers in knowledge-intensive domains such as medicine and engineering: learning by watching an expert solve problems, learning by watching a student solve problems, and learning by watching oneself solve problems. In the first scenario, which is called *learning by watching an expert*, a student apprentice observes the problem-solving actions of a human expert. A learning opportunity occurs when the student cannot explain an observed action. At this point, the problem-solving context often allows the student to conjecture the knowledge discrepancy responsible for the explanation failure. The conjecture can be evaluated by the use of an underlying theory of the domain or by asking the expert. In this thesis, the 'student' in the scenario that was just described is an expert system to be improved; the ODYSSEUS apprenticeship learning program allows the expert system to perform the learning method used by the student apprentice. The major objective this

chapter is to provide a detailed introduction to this first learning by watching scenario, where the student to be improved is an expert system.

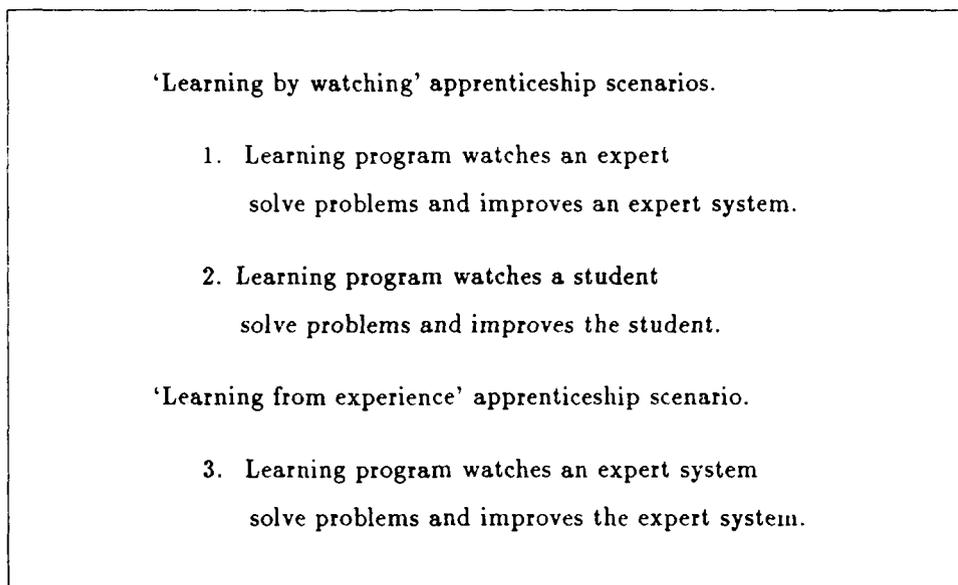


Figure 1.2: The three apprenticeship scenarios that are investigated in this thesis. The principal focus of this chapter and thesis is the first of the three apprenticeship scenarios.

The second principal apprenticeship learning scenario is *learning by watching a student*, and refers to the situation when a human expert watches and critiques the problem-solving behavior of a student apprentice. In this thesis, the 'student' in the scenario that was just described is a human student to be improved; the ODYSSEUS apprenticeship learning program allows the expert system to perform the learning method used by the expert. Figure 1.1 illustrates the experimental set-up used to investigate the two learning by watching scenarios that have been described. ODYSSEUS uses the exact same mechanisms to automate the transfer of expertise in both situations, so the term *learning by watching* will often be used to refer to both of them.

The third apprenticeship learning scenario involves watching ones own problem solving, and this scenario is referred to as *learning from experience*. A student apprentice

learns in the course of solving problems and monitoring his or her own problem-solving failures. In the experimental setup used to investigate this scenario, an expert system assumes the role of the student doing problem solving. The learning program detects when the meta-level strategy knowledge of the expert system fails to achieve problem-solving goals. The ODYSSEUS learning program then suggests and evaluates repairs to the object-level knowledge base that would prevent the meta-level problem-solving failures.

The ODYSSEUS learning apprentice is designed to improve any knowledge base crafted for an expert system shell called HERACLES.<sup>1</sup> HERACLES uses a problem-solving method called heuristic classification, which is the process of selecting a solution out of a pre-enumerated solution set, using heuristic techniques (Clancey, 1985). In a *knowledge acquisition* setting, ODYSSEUS improves the object-level knowledge of an expert system in the process of watching a human expert or the expert system solve problems, and thereby functions as a knowledge acquisition program for HERACLES. In an *intelligent tutoring* setting, ODYSSEUS improves the object-level knowledge of a student in the process of watching and critiquing a student who is solving problems, and thereby functions as a student modeling program for the GUIDON2 intelligent tutoring system, which is built over HERACLES.

## 1.2 Odysseus Apprenticeship Learning Program

### 1.2.1 The Performance Model of Learning

Having defined the apprenticeship situation, a careful look will now be given to the

---

<sup>1</sup> HERACLES is described at length in Section 2.1 and Appendix B. The best single reference is (Clancey and Bock, 1986). HERACLES was created by removing the medical knowledge from NEOMYCIN (Clancey, 1984). In this thesis, there is no meaningful distinction between the use of the terms HERACLES and NEOMYCIN, because our experiments used HERACLES with the NEOMYCIN medical knowledge base.

tasks that are faced by an apprenticeship learner within the context of the performance learning model (Buchanan et al., 1978) shown in Figure 1.3, and then outline the solution approach presented in this thesis. The major functional components of the performance learning model are a performance element, a training instances generator, an intelligent editor and a learning critic. Each of these will be described in turn.

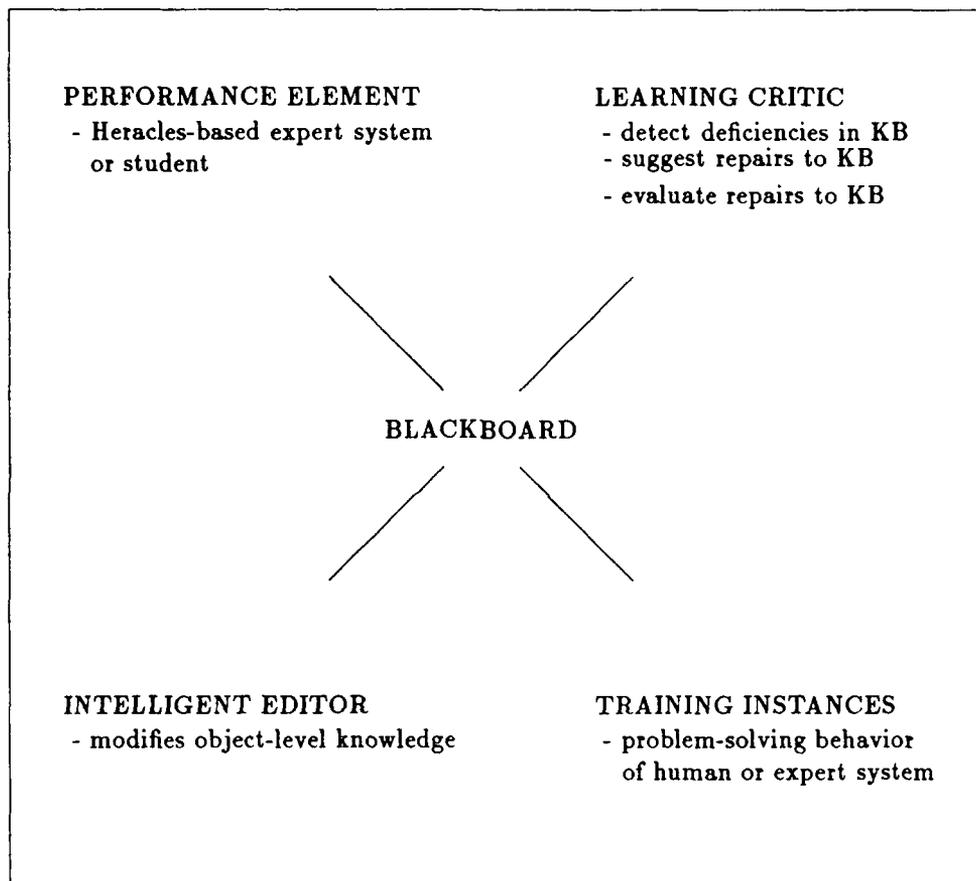


Figure 1.3: The performance model of learning provides a good framework for describing learning programs. The components of our apprenticeship learning situation are shown within this model.

The term *performance element* refers to the system that the learning program improves. In a knowledge acquisition setting, the performance element is an expert system written in the HERACLES shell. In a intelligent tutoring system setting, the performance element is a student. More generally, the techniques described in this thesis are applicable

to any knowledge based system where domain-specific knowledge is described at an object-level and strategy knowledge of the problem-solving method is represented explicitly at a meta-level.

The *training instances* presented to the learning system consist of the normal observable problem-solving behavior of an expert or expert system, when the performance element to be improved is the expert system. In this thesis, training instances will also be referred to as *data requests* or *actions*. The behavior of a student is observed when the performance element to be improved is a student. In our experiments, the human problem solver is given a chief complaint of a patient, and then proceeds to make a series of data requests to gather information that allows the patient's illness to be determined. An example is shown in Figure 1.5. The problem solver answers any clarification requests made by the learning system critic, relating to goal information. For example, the problem solver's goal might be to confirm a particular hypothesis or to discriminate between two likely hypotheses.

The heart of the learning system is the *learning critic*, which is responsible for global and local credit assignment. An apprentice learner directly confronts these two problems, which are two of the hardest and most central problems in machine learning (Dietterich and Buchanan, 1981). The *global credit assignment* problem is to determine when the observable behavior of an human problem solver suggests a knowledge base deficiency. In an apprenticeship learning setting, the function of the *global critic* is to answer a yes or no question: does an observed action of the problem solver suggest a knowledge base deficiency in the expert system? Since a 'yes' answer relates a knowledge base deficiency to a particular action of the specialist, the global critic significantly contributes to localizing the knowledge base deficiency. The *local credit assignment* problem is to isolate the deficiency, which is very difficult in a complex knowledge-based program.

In this thesis, the global and local credit assignment problems are divided into three distinct problems, and each of them is attacked separately. The three problems are to detect discrepancies in the knowledge base, to suggest knowledge base repairs, and to evaluate suggested knowledge base repairs. Any learning requires solving these three problems. In most existing knowledge acquisition programs, these three functions are not performed automatically; a cooperating human expert assists the learning program in solving these problems. The research described herein provides a method of automating all three of these tasks.

The *intelligent editor* can reason about the performance element's knowledge structures. In our case, this means it knows how to modify the HERACLES domain-specific parameter and rule files, thereby correcting bugs or adding or deleting new object-level knowledge.

### 1.2.2 Odysseus' Learning Methodology

The solution approach that ODYSSEUS takes to a learning by watching situation is shown in Figure 1.4. The purpose of the remainder of this section provides an informal introduction to the ODYSSEUS method. In Chapters 2-5, the solution approach will be explained in a more technical fashion.

The top third of the figure illustrates how ODYSSEUS accomplishes the detection of deficiencies. Each time an human problem solver performs a problem-solving action, the action is input to ODYSSEUS. The ODYSSEUS explanation generator produces explanations of the human's action. These explanations are sequences of strategy metarules that connect the observed action to a high-level problem-solving goal. They can be viewed as PROLOG-style proof trees; the leaves of the proof trees are instantiated using the object-level knowledge base. When no proof tree can be successfully instantiated, an

explanation failure occurs. The output of the program that detects deficiencies is the unexplained action of the human expert and a list of the active high-level goals to which ODYSSEUS believes the action most likely relates. This is passed to the next learning stage of ODYSSEUS, which suggests repairs.

The ODYSSEUS subsystem that suggests repairs to the knowledge base relaxes the constraints on the construction of explanatory proof trees, beginning with a single fault assumption. That is, it constructs PROLOG-type proof trees that succeed in connecting the unexplained action to a high-level goal when a single clause in one of the premises of metarules forming the explanation chain is allowed to fail. Examples of two such clauses are `subsumes($parameter1 $parameter2)` and `evidence.for($parm1 $hypothesis1 $rule1 $cf1)`. It then conjectures which knowledge is missing from the knowledge base by generating all instantiations of the failed clause that are consistent with the problem-solving context. The problem-solving context reduces the allowable values of the variables in the clause. Each time a suggestion is produced, it is passed to the third stage of ODYSSEUS that evaluates candidate repairs.

The subsystem that evaluates repairs determines the worth of the repair through the use of a confirmation theory. A confirmation theory is a decision procedure that is given an arbitrary instantiation of a clause and determines if it should be added to the knowledge base. The ODYSSEUS confirmation theory includes a set of constraints for each type of clause, which must be met if the clause is to be added to the knowledge. In trying to satisfy these constraints, the confirmation theory uses the existing knowledge base and the underlying domain theory for the knowledge base. When a candidate repair is accepted by the confirmation theory, the clause is added to the object-level knowledge base.

The solution approach used by ODYSSEUS in the third apprenticeship learning sce-

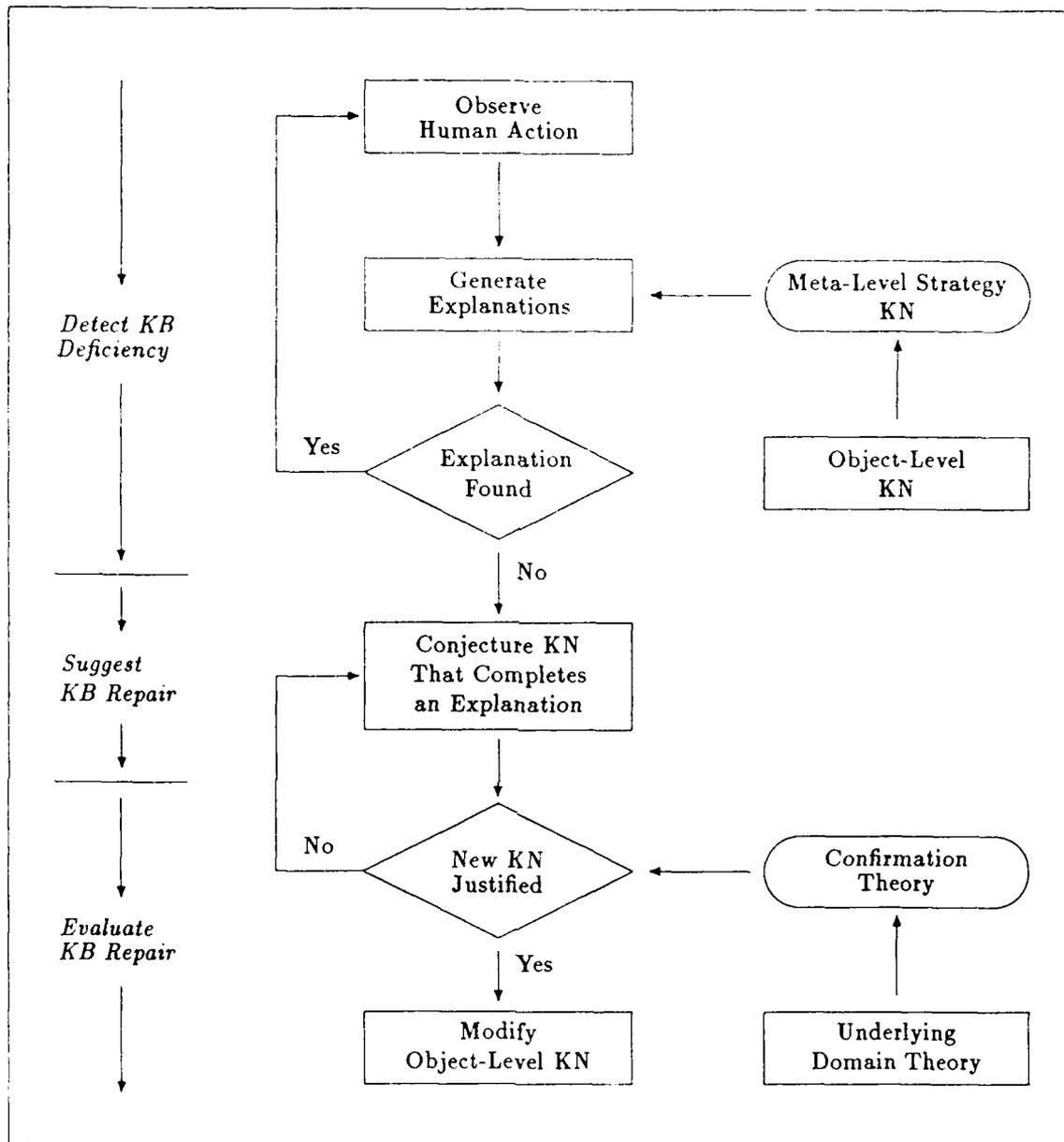


Figure 1.4: Overview of ODYSSEUS' method in a learning by watching apprentice situation. This thesis develops techniques that permit automation of each of the three stages of learning shown on the left edge of the figure. An explanation consists of a sequence of metarules that connect a problem-solving action to a high-level goal.

nario, called learning from experience, will now be described. In this scenario, the learning program monitors the normal problem solving of a HERACLES-based expert system. Once again the learning method has three stages. The first stage detects possible deficiencies by

monitoring failures of the premises of strategy metarules in the expert system shell, since all knowledge base gaps cause these. The exact same techniques used for the last two stages of learning by watching are also applicable to learning from experience. Indeed, the second stage is easier in learning from experience, since the input includes the particular clause and the particular metarule that failed. In learning by watching these must be determined from more general information.

### **1.3 Example of Learning by Watching**

An example in the domain of medical diagnosis will illustrate how ODYSSEUS improves an expert system using apprenticeship techniques in a learning by watching an expert scenario. This example is meant to convey an intuitive understanding of our solution approach. This same example is treated in greater depth in Chapters 3-5. Each of these three chapters expounds on one of the three stages of apprenticeship learning.

A typical training instance that is input to the learning program is the problem-solving behavior of an expert physician, such as shown in Figure 1.5. The expert begins making data requests and concludes with a final diagnosis. ODYSSEUS has no explanation for why this data request was asked and thus a discrepancy has been detected.

#### **1.3.1 Stage 1: Detecting Deficiencies in a Knowledge Base**

The first of the three stages of apprenticeship learning is detection of knowledge base deficiencies. For each action of the expert, ODYSSEUS generates one or more explanations of the expert's behavior. These are referred to as Line of Reasoning Explanations (LOREs). They consist of a sequence of strategic metarules that connect an observed action of the problem solver to a high-level strategic goal, in which free variables in the strategic metarules are instantiated with object-level knowledge. A learning opportunity occurs

**Patient's Complaint and Volunteered Information:**

1. Alice Ecila, a 41 year old black female.
2. Chief complaint is a headache.

**Physician's Data Requests:**

3. Headache duration? focus=tension headache. 7 days.
4. Headache episodic? focus=tension headache. No.
5. Headache severity? focus=tension headache. 4 (Scale 0-4).
6. Visual problems? focus=subarachnoid hemorrhage (sah). Yes.
7. Double vision? focus=sah, tumor. Yes.
8. Temperature? focus=infectious process. 98.7 Fahrenheit.  
.....
14. Visual sensitivity to light? focus=migraine, viral meningitis. Yes.
15. Sickdegree? focus=viral meningitis, infection, sah, tumor. 1 (Scale 0-4).
16. Stiff Neck? focus=meningitis. No.
17. Nausea? focus=migraine. Yes.
18. Vomiting? focus=migraine. Yes.
19. Red and painful eye? focus=cluster headache. No.  
.....

**Physician's Final Diagnosis:**

25. Migraine Headache.

Figure 1.5: An example of what the ODYSSEUS apprentice learner sees. The data requests in this problem-solving protocol were made by John Sotos, M.D. The answers to his data requests were obtained from an actual patient file from the Stanford University Hospital, extracted by Edward Herskovits, M.D.

when no LORE can be found for an observed action. In the current example, no explanation can be found for one of the expert's actions, namely asking the question visual problems.

The program announces this fact.

### 1.3.2 Stage 2: Suggesting Repairs to a Knowledge Base

The second stage of apprenticeship learning is to suggest repairs to the knowledge

base. ODYSSEUS relaxes the constraints on the generation of LOREs. LOREs are constructed that contain metarules with one or more failed metarule clauses; initially, only a single failed clause is allowed. In this example, one of the LOREs that is constructed contains the failed metarule premise clause `evidence.for($finding $hypothesis $rule.name $cf)`. To suggest repairs, ODYSSEUS finds all sets of allowable bindings for the variables in this clause. These bindings are constrained by the bindings of the variables in other clauses of the metarules, but they are not constrained by which `evidence.for` tuples are actually in the knowledge base. An example of a metarule clause suggested by this technique is `evidence.for(photophobia acute.meningitis $rule1 $cf)`. The candidate suggests that if a rule was added to the knowledge base that mentions the finding `photophobia` in the premise and provides evidence for `acute meningitis` in its conclusion, then a LORE explanation could be completed that explained the data request `visual problems`.

### 1.3.3 Stage 3: Evaluating Repairs to a Knowledge Base

The third step of apprenticeship learning is to evaluate the candidate repairs using a confirmation theory. Our confirmation theory consists of a decision procedure for each relation, such as `evidence.for`. Each clause of a metarule provides an example of one or more relations. Candidate repairs are tested to see whether they satisfy the decision procedure for their relation. In reaching a decision, the decision procedure uses underlying theories of the domain and the existing object-level knowledge base. The underlying theory of the domain used by ODYSSEUS includes a LORE explanation generator based on an explicit representation of strategy knowledge for heuristic classification, and an induction system that uses a library of correctly solved cases and a set of rule 'goodness' measures that determine the quality of an arbitrary rule presented to it. In the example that has been developed, the induction system generates and evaluates rules that have `photophobia` in

their premise and acute meningitis in their conclusion. A rule is found that passes the rule 'goodness' measures, and is automatically added to the object-level knowledge base.

This completes our informal example of the method used by ODYSSEUS in a learning by watching apprenticeship. This same example is developed in greater depth in Chapters 3-5.

## **1.4 Reader's Guide**

This chapter provided an overview of the entire thesis. The ODYSSEUS learning method was described and an example of learning by watching was presented. The next chapter provides background information. It describes the method of knowledge representation and control used by the performance element to be improved, and surveys the past research that relates to the apprenticeship learning problem.

A suite of three chapters then describes the three stages of apprenticeship learning. These are followed by a second suite of three chapters, describing the inherent, upper, and lower limits of apprenticeship learning, respectively. Finally, we summarize.

## CHAPTER 2

### BACKGROUND

The problem solving architecture of the performance element and the syntax and semantics of the knowledge base have a major impact on a learning method. This chapter describes the method of knowledge representation and control used by the performance element. This chapter also provides an overview of past research that relates to the apprenticeship learning techniques that are developed in this thesis.

#### 2.1 Problem-Solving Architecture of Heracles

The implementation of the ODYSSEUS learning apprentice program works in conjunction with the HERACLES expert system shell. Understanding the architecture of HERACLES is important for understanding the learning techniques used by ODYSSEUS. The learning techniques developed in this thesis should be generally applicable to any performance element that has an explicit meta-level representation of strategy knowledge.

HERACLES solves problems using the heuristic classification method (Clancey, 1985). This method selects a member of a pre-enumerated solution set using heuristic methods. EMYCIN is another example of an expert system shell that solves problems using the heuristic classification method (Buchanan and Shortliffe, 1984). HERACLES was

created by removing the medical knowledge from NEOMYCIN (Clancey, 1984). HERACLES is to NEOMYCIN as EMYCIN is to MYCIN (Buchanan and Shortliffe, 1984). The relationship between these programs is illustrated in Appendix D. The reader desiring details of HERACLES beyond what is provided in this chapter is referred to Appendix C.

Problem-solving knowledge in HERACLES is organized into three distinct layers, as illustrated in Figure 2.1. The bottom object-level of organization includes all domain-specific knowledge of the expert domain, such as medical or engineering knowledge. The middle meta-level layer contains strategy knowledge for a class of problems, such as heuristic classification or constraint propagation. Earlier shells such as EMYCIN did not have the middle layer of strategy knowledge; rather, this knowledge was imbedded in the interpreter (top level) and the domain rules (bottom level). The top layer of Heracles is a task interpreter which interprets the strategy knowledge, which is represented as tasks and metarules. The remainder of Section 2.1 describes these three layers in greater detail.

### 2.1.1 Object-Level Representation

Two characteristics of the object-level representation facilitate learning: distinctions are made between the different types of domain knowledge in the knowledge base, and all rule and frame knowledge at the object-level is uniformly encoded as a database of ground literals.

### 2.1.2 Syntax of Object-Level Knowledge

Object-level knowledge has an extensional and intensional component. The extensional data base (EDB) is created by instantiating a set of predefined *relations*  $R$ , specific to the problem domain. The EDB has a *relation schema*, a list of the attributes of each relation (e.g.,  $R(A, B, C)$ ). The relation schema of NEOMYCIN is shown in Appendix C.

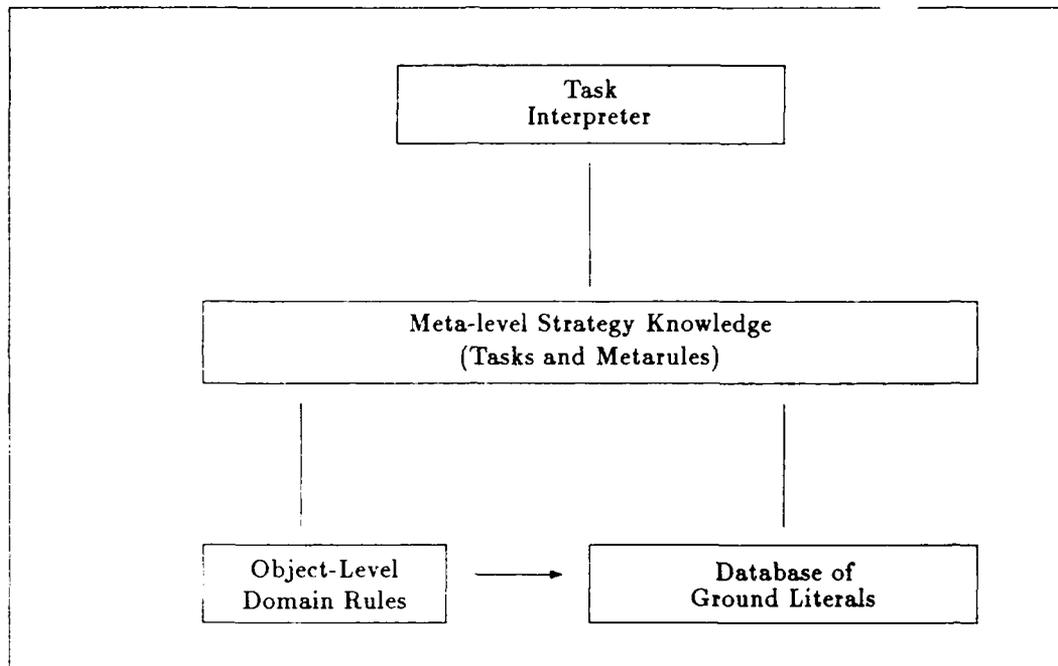


Figure 2.1: HERACLES problem solving architecture. The middle level encodes strategy knowledge of a problem-solving method, such as diagnosis. The bottom level encodes domain-specific knowledge, such as medical knowledge.

The permissible ranges of the values of the attributes of the schema are known. Following the database literature, we shall refer to instantiated relations as *relation instances*, which are sets of *tuples*. A tuple has the form  $R(c_1, c_2, \dots, c_n)$ , where  $c_1$  through  $c_n$  are in the set of legal values for their respective attributes, and  $R$  is a relation name.

The relation schema for NEOMYCIN contains 120 different relations. Many different types of knowledge are encoded, such as hierarchical, associational, heuristic, control, definitional, and causal knowledge. Even rule knowledge is represented in terms of relations; a single rule is decomposed into tuples in many different relations. The actual representation follows the MRS logic programming language (Russell, 1985) conventions for encoding tuples. For example, an instantiation of the proposition `suggests($parameter $hypothesis)` represents the fact that if a particular parameter is true then this suggests

that a particular hypothesis is true. An instantiation of the template `askfirst($parameter $flag-value)` specifies whether the system should first ask the user for the value of a finding, or derive the information from existing information.

The intensional data base (IDB) at the object-level consists of rules with a MYCIN-like syntax. The premise and action of these rules initially each consist of a conjunction of tuples. To make this knowledge more declarative, each rule is translated into a set of tuples in the EDB as described in the following section.

### 2.1.3 Specification of Object-Level Knowledge

Domain-specific knowledge in HERACLES is initially specified by the user as MYCIN-like rules and facts. This knowledge is then translated into the knowledge representation constructs of the MRS logic programming language. For clarity, a hypothetical example of this translation process will be shown. An original representation as objects and rules is as follows.

```

headache.duration
    parmtime soft.data
    subsumed.by headache
    askfirst T
    ...

rule160
    premise (and (equal headache-chronicity acute)
                (equal headache-onset abrupt)
                (greater (value headache-severity) 3))
    action (conclude sub-hemorrhage yes .6)

```

```

trigger.rule T
antecedent.rule T
...

```

This knowledge is translated by Heracles into MRS, reducing it to a database of ground literals. For the current example, the result of this translation is:

```

parmtype(headache.duration soft.data)
subsumes(headache.duration headache)
askfirst(headache.duration)
evidence.for(headache.duration sub-hemorrhage rule160 .6)
evidence.for(headache.onset sub-hemorrhage rule160 .6)
evidence.for(headache.severity sub-hemorrhage rule160 .6)
antecedent(headache.duration rule160)
antecedent(headache.onset rule160)
antecedent(headache.severity rule160)
trigger.rule(rule160, headache-severity)
antecedent.rule(rule160)

```

This translation shows a few of the approximately 120 knowledge relations defined in Heracles, such as `subsumes($parm1 $parm2)`, `evidence.for($parameter $hypothesis $rule $cf)`, and `trigger($rule)`. The `evidence.for` tuple means that `$parameter` contributes evidence for `$hypothesis` in the rule named `$rule` and the certainty factor or strength of this rule is `$cf`. A '\$' denotes a variable. If a rule has several parameters in the premise, an `evidence.for` tuple is constructed for each of them. The goal of such a translation is to reduce the domain-knowledge to a completely declarative representation. In the original representation, knowledge is procedurally imbedded, for example the ordering of clauses in the rules premise.

The EDB representation for a heuristic rule that contains a conjunction loses some of the information in the original representation. For example, the translation does not distinguish between the rule  $a \wedge b \wedge c \wedge \text{---} d$  (cf1) and the rule  $a \wedge \neg b \wedge c \wedge \text{---} d$  (cf1). The original set of rules are still maintained in the system; they are executed and change the state of the system as a side-effect.

#### 2.1.4 Semantics of Object-Level Knowledge

ODYSSEUS contains a confirmation theory consisting of a decision procedure for relations. The confirmation theory is implemented in LISP. This allows candidate knowledge base repairs, in the form of tuples, to be tested for quality before being added to the knowledge base.

Our confirmation theory is a partial theory with respect to the vocabulary of relations at this point: decision procedures have not been constructed for all relations. Inputs to decision procedures, include the candidate repair, the ODYSSEUS induction program, a set of correctly solved cases in the domain, the current knowledge base, the ODYSSEUS explanation generator, and NEOMYCIN's explicit model of strategy reasoning.

#### 2.1.5 Meta-Level Representation

In HERACLES, control knowledge is represented as tasks and metarules, which are invoked by a forward-chaining *task interpreter*. A *task* is a procedure for accomplishing some well-defined problem-solving subgoal. Examples of tasks are to test a hypothesis, group and differentiate hypotheses, refine a hypothesis, forward reason, ask general questions, and process hard data.

Each method within a task procedure for achieving the task goal of the procedure is called a *metarule*. Metarules do not contain object-level knowledge; they locate and apply object-level knowledge by content-directed invocation (Davis, 1982). The metarule

premise consists of a conjunction of uninstantiated EDB clauses,  $c_1 \wedge c_2 \wedge \dots \wedge c_n$ . The EDB includes tuples that encode dynamic problem-solving state information. The task interpreter determines if the set of EDB tuples allows the clauses of the metarules to be unified. The bindings for variables in the premise are passed to the action part of the metarule. The action of a metarule performs one or more of the following operations: call another task (i.e., invoke a subgoal), request information from the user, modify dynamic state information, or apply an object-level rule.

Tasks and metarules can be viewed as orchestrating the object-level knowledge: they piece together it in order to achieve a problem-solving goal. Currently HERACLES contains approximately thirty task procedures and sixty metarules.

## 2.2 Related Research

There is a large body of past research that shares ODYSSEUS' goal of automating the transfer of expertise. There is also a large body of past research that employs ODYSSEUS' technique of constructing explanations or plans to explain observed problem-solving behavior. Thus our coverage of related research includes the subfields of knowledge acquisition, machine learning, intelligent tutoring, plan recognition, and natural language understanding.

### 2.2.1 Knowledge Acquisition

There are seven outstanding existing AI programs for semi-automating end-game knowledge acquisition for expert systems: TEIRESIAS, SEEK2, INDUCE, ID3, RL, MORE and LEAP. Each of these will be discussed in turn. Our discussion will emphasize the extent to which they automate the three stages of learning: detecting deficiencies, suggesting repairs, and validating repairs.

TEIRESIAS is a knowledge acquisition program for the MYCIN expert system that pioneered research in knowledge acquisition for expert systems (Buchanan and Shortliffe, 1984; Davis, 1982). However, the difficult problem of detecting knowledge deficiencies is circumvented by TEIRESIAS; it requires the use of a human expert who watches an EMYCIN-based expert system solve problems, and who tells TEIRESIAS when a knowledge base deficiency exists. Upon receiving this signal, TEIRESIAS unwinds the reasoning chain relating to the questionable action of the expert system under the guidance of a human expert. The expert tells TEIRESIAS which rule to modify and how it should be modified, and thus also plays a major role in suggesting and validating the repair.

SEEK2 aids knowledge base refinement of the EXPERT expert system (Ginsberg et al., 1985; Politakis and Weiss, 1984). SEEK2 doesn't really debug a knowledge base; rather, it 'tweaks' the existing heuristic rules to optimize problem-solving performance, using modification operators such as LOWER-CF and RAISE-CF (Ginsberg, 1986). We have argued that that raising or lowering the certainty factors of rules, when those certainty factors have been calculated using a representative case library, is without scientific foundation and should not be practiced (Wilkins and Buchanan, 1986). When a representative set of past cases is present, the strengths of inexact rules are determined, since certainty factors can be given a strict probabilistic interpretation (Heckerman, 1986). Knowledge bases are increasingly used for multiple purposes. Modifying correct object-level knowledge to be incorrect, so that performance is improved for a particular function such as diagnosis, lessens the likelihood that the same knowledge base can be used for other functions such as design, explanation, and teaching.

INDUCE ID3, and RLcreate rule bases by induction over a case library (Michalski, 1985; Quinlan, 1983; Fu and Buchanan, 1985). For these programs and SEEK2, a training instance is an entire problem-solving session, not a single problem-solving action of an

expert as in ODYSSEUS. Hence, the training instances used by ODYSSEUS are finer-grained, and can exploit the sequence information implicit in problem-solving protocols. More importantly, the performance element associated with these four systems do not contain strategy knowledge that attempts to mimic the problem-solving behavior of human experts. Such strategy knowledge is important for ODYSSEUS, because it provides a basis for following the line of reasoning of an expert, a key subtask in accomplishing fine-grained deficiency detection.

MORE is an interactive manual method for improving the knowledge base of a classification expert system (Kahn et al., 1985). The MORE program analyzes the form of the knowledge base off-line, to see whether the knowledge base satisfies a set of pre-specified constraints. The set of pre-specified constraints is somewhat analogous to a bug library. One such constraint might be that every critical finding, such as a temperature of 105 degrees Fahrenheit, must relate to at least one disease. If this is found not to be the case, then MORE would inform the expert of this gap in the knowledge base. MORE is interesting in that it automates the detection of deficiencies. Suggesting and evaluating repairs is left to the human expert.

The LEAP learning system for the VEXED VLSI circuit design aid, currently under development, is a noteworthy example of an apprentice learning system for a knowledge-intensive task (Mitchell et al., 1985). The first two stages of apprenticeship learning, the detection of deficiencies and the suggesting of repairs, are the responsibility of the human expert who is using the VEXED circuit design aid. LEAP evaluates this repair using circuit theory as an underlying theory of the domain, and then generalizes this repair.

It will be interesting to see if the LEAP approach is successful at refining the knowledge base for the VEXED circuit design aid, because the LEAP approach makes two assumptions that we suspect are false. First, LEAP assumes that it is unlikely to have

deleterious interactions between rules of the knowledge base, and our experience leads us to believe that all knowledge-based expert systems will have interactions between elements. Second, LEAP makes the assumption that the entire knowledge base can be supported by a strong underlying domain theory, i.e., one that is capable of deductively proving the validity of all elements of the knowledge base in a logical sense. We suspect that all real-world systems have knowledge that cannot be supported in this fashion. Control knowledge seems particularly resistant to this type of underpinning.

### 2.2.2 Machine Learning

Some of the earliest work in machine learning was directed at developing learning by watching programs in the areas of games and mathematics. Examples include Samuel's checker program (Samuel, 1963), Waterman's poker player (Waterman, 1970), and Mitchell's LEX program (Mitchell et al., 1983). In such domains, the detection of deficiencies and the evaluation of repairs is usually quite tractable. Generally, this research has not been of relevance to learning for expert system knowledge bases, because all object-level knowledge is given at the start, and only the strategy knowledge for applying the object-level knowledge is learned.

The problem solver used by Samuel's program was a book of championship checker games. The detection of deficiencies was accomplished by seeing if the book move was the same as the move that Samuel's checker program makes in the same situation. Suggesting and validating repairs involved adjusting the coefficients of a polynomial evaluation function for the selection of moves.

Waterman's exemplary programming system is directed at synthesizing an algorithmic activity by observing multiple instances of its application (Waterman, 1978). This method seems to work for synthesizing small programs such as performing a file transfer

between two computer systems.

Another program that learns in an apprenticeship context is the PRE system for theory-directed data interpretation (Dietterich, 1984). PRE learns programs for Unix commands from examples of the use of the commands. The learning program employs constraint propagation to identify differences between the problem solver and the performance program for commands.

The field of machine learning is undergoing a renaissance (Michalski et al., 1983; Michalski et al., 1986; Mitchell et al., 1986) and at the center of this renaissance is *explanation based learning* (EBL), also called *explanation based generalization* (EBG) (Mitchell et al., 1986; DeJong and Mooney, 1986). EBL/EBG can be defined as the ability to acquire new knowledge through intensive analysis of a *single learning example* (also called a training instance) using a *domain theory*. The major differences between EBL/EBG research and ODYSSEUS relate to the use of explanation structures and the use of domain theories.

The creation of explanation structures is very central to EBL/EBG and ODYSSEUS. For both of these efforts, the leaves of the explanation structure proof trees bind to tuples of the domain theory. In ODYSSEUS, the domain theory is the object-level knowledge base of an expert system. In EBL/EBG, a learning opportunity occurs when an explanation structure is constructed using the domain theory. ODYSSEUS learning can only take place when an explanation structure cannot be constructed from the domain theory. In EBL/EBG, the domain theory is used to learn or operationalize concepts. In ODYSSEUS, improvement to the domain theory, itself, is the objective of learning.

EBL/EBG techniques require that the domain theory be capable of *deductively proving* that the learning example is an instance of the concept to be learned. Human expertise often involves inexactness, uncertainty, and the use of heuristics; hence there is no way in principle for a human expert or expert system to deductively prove their

conclusions in knowledge-intensive domains.

### 2.2.3 Student Modeling

A class of systems very closely related to apprenticeship learning has been developed in the area of intelligent tutoring. Instead of an apprentice watching an expert, these systems involve an expert watching an apprentice during normal problem solving. Intelligent tutoring research is very relevant to the apprenticeship learning approach we are taking. The chief skill needed in the detection of knowledge base deficiencies is the ability to follow the line of reasoning of a human problem solver; this is achieved by a comparison of the actions of the problem solver to the prediction of a program that also knows how to solve the problem. There has been excellent research in a formal mathematical domain that focused on following the line of reasoning of a human problem solver and that emphasized the importance of having a cognitively plausible performance element. Examples include DEBUGGY (Brown and Burton, 1978), Repair and Step Theory (Brown and VanLehn, 1980; VanLehn, 1983), DPF and ACM (Ohlsson and Langley, 1985; Langley et al., 1984), LMS (Sleeman and Brown, 1981), ACT (Anderson, 1983), and the MACSYMA-ADVISOR (Genesereth, 1981). Other research in a formal games domain includes the WHY (Collins et al., 1987), WEST (Burton and Brown, 1982) and WUMPUS (Goldstein, 1978) programs. This activity of following the line of reasoning goes under many names, such as cognitive diagnosis, student modeling, model tracing, and automated protocol analysis.

In intelligent tutoring the goal is to 'debug' a human problem solver. Many intelligent tutoring systems contain a performance element that is capable of solving the same problem that is given to the student and model the student against the performance element, including the WEST program in the domain of games (Burton and Brown, 1982), SOPHIE III and GUIDON in the domain of diagnosis (Brown et al., 1982; Clancey, 1979), and

MACSYMA-ADVISOR in the domain of symbolic integration (Genesereth, 1982). There are many similarities between MACSYMA-ADVISOR and ODYSSEUS. Both take as input a sequence of actions of a problem solver, and try to recognize the plan of the problem solver. SOPHIE III uses an expert system for circuit diagnosis as an aid in isolating hypothesis errors in the behavior of students who are performing electronic troubleshooting. GUIDON is built over the MYCIN expert system for medical diagnosis (Buchanan and Shortliffe, 1984); student hypothesis errors are discovered in the process of conducting a Socratic dialogue.

There are interesting parallels between VanLehn's SIERRA system (VanLehn, 1983) and the ODYSSEUS system. Both learning systems try to explain the observed problem-solving behavior, using different methods of completing explanations. SIERRA's domain is mathematical subtraction, so the global credit assignment is trivial: a computer simply needs to check whether the problem solver subtracted correctly. In conjecturing repairs, both SIERRA and ODYSSEUS begin with the single fault assumption. SIERRA allows one new subprocedure while ODYSSEUS allows one new declarative domain fact. The principal difference between them is that SIERRA's focus is learning procedural knowledge while ODYSSEUS' focus is learning declarative factual knowledge.

There are three reasons why this intelligent tutoring research fails to solve the problems faced by ODYSSEUS in automating knowledge acquisition for expert systems. First, most work uses a bug library to determine differences between a model of the problem domain and the behavior of a human problem solver. The use of a bug library destroys the symmetry between learning and teaching, and thus makes this work inapplicable. Second, it is carried out in formal domains such as mathematics and game worlds that introduce simplifying assumptions not present in the informal domains in which some expert systems operate. Third, it focuses on learning procedural knowledge and detecting proce-

dural errors, and ODYSSEUS focuses on learning and detecting deficiencies in declarative object-level knowledge. Nevertheless, the intelligent tutoring work serves as a source of inspiration, and our work can rightfully be viewed as an attempt to extend this previous work into informal domains where the knowledge to be refined and debugged is declarative object-level knowledge.

#### 2.2.4 Neomycin-based Modeling Programs

In addition to ODYSSEUS there are two other modeling programs currently under construction within the NEOMYCIN framework that relate to the transfer of expertise. This section describes the differences between these efforts and ODYSSEUS.

Whereas ODYSSEUS attempts to detect knowledge base discrepancies in a knowledge acquisition setting by following the line-of-reasoning of a problem solver, the GUIDON-DEBUG program detects discrepancies by analysis of the *problem solution* that is constructed when the NEOMYCIN expert systems solves a problem (Clancey, 1986; Clancey, 1987). GUIDON-DEBUG analyzes the problem solution to determine whether it satisfies a set of pre-specified constraints. For example, in a medical diagnosis domain, certain findings are marked as critical findings, such as a temperature of 105 degrees Fahrenheit. A pre-specified constraint on critical findings used by GUIDON-DEBUG is that the final diagnosis must explain the critical findings. So GUIDON-DEBUG checks to see whether the final diagnosis explains this critical finding, and if not, the expert is informed that there is an error in the expert system knowledge base. The GUIDON-DEBUG approach is most similar to the MORE system for knowledge acquisition. Like MORE, GUIDON-DEBUG addresses the problem of detecting and pinpointing areas of the knowledge base where there are deficiencies. However, neither GUIDON-DEBUG nor MORE generate the space of potential repairs or evaluate the correctness of repairs.

GUIDON-DEBUG is also useful for intelligent tutoring: a student watches the program produce an evolving representation of the solution of the expert system. This evolving solution is called a Patient Specific Model, a concept developed in ABEL (Patil et al., 1981). The hypothesis under investigation in GUIDON-DEBUG is that the student's strategic reasoning will improve by being exposed to a display of NEOMYCIN's strategic reasoning. Editing the patient specific model provides the student with an opportunity to debug the NEOMYCIN knowledge base.

The IMAGE program is another modeling program that uses the NEOMYCIN expert system as a foundation (London and Clancey, 1982). Whereas, the goal of ODYSSEUS is to detect discrepancies in the object-level knowledge, the goal of IMAGE is to identify the meta-level strategy being used by the student problem solver, and detect when the meta-level strategy of a student is different from that of NEOMYCIN. Unlike ODYSSEUS, IMAGE is only applicable to student modeling; it is not applicable to knowledge acquisition, i.e., the learning of new strategy metarules for NEOMYCIN. The use of an underlying domain theory is not relevant to IMAGE.

IMAGE faces a very difficult task. Following a strategy of a student becomes almost impossible if the medical domain knowledge of the student is much different from the NEOMYCIN expert system. Protocol experiments and the use of the ODYSSEUS explanation generator have revealed that a student problem solver does have much different domain knowledge than an expert and expert system. Free-form protocol experiments of second-year medical students were collected by by Bill Clancey, Curt Kapsner, M.D., Bob London, and David Wilkins. These were processed by the ODYSSEUS LORE generator, and this revealed that the knowledge used by a student in solving problems in the NEOMYCIN domain is not in NEOMYCIN on 75% of the data requests made by the student. This means that there is a *cognitive tear* in the model produced by the student

modeler on 75% of the observed actions of the student. The concept of a cognitive tear is due to Brown and Burton and refers to the problem a student modeler faces when noise masquerades as data (Burton and Brown, 1982). Not only does a modeler lack a basis for producing a correct interpretation on 75% of the data requests, but these data request will most probably lead the strategy modeler astray in producing the correct interpretation on the remaining 25% of the data requests, where the correct interpretation can in principle be determined. Based on this result, we conjecture that it is impossible in principle for IMAGE or *any* modeling program to follow the strategic reasoning of a student, when using the NEOMYCIN knowledge base.

### 2.2.5 Plan Recognition and Language Understanding

There has been a great deal of research on failure-driven learning that monitors control and planning knowledge (Mitchell et al., 1983; Korf, 1985; Minton, 1985). The goal of these research efforts is to create better control knowledge so as to speed up problem solving, rather than to learn domain-specific factual knowledge. This compliments our approach, as we do not address the learning of control knowledge for a problem-solving method; in other words, we do not learn Heracles strategic tasks and metarules.

The ODYSSEUS approach has strong similarities to work in natural language understanding (Charniak, 1977; Dyer, 1973). This research tries to determine the reasons for and the assumptions in stories. The work on memory organization packets (MOPs) proposed by Schank provides a technique for failure-driven learning (Schank, 1982). These systems perform generalizations to modify their memory structures just enough to account for new information. The precondition analysis method of the Silver's LP program for learning algebraic equation solving is also relevant, because of its emphasis on the use of meta-level reasoning (Silver, 1986). However, in this work and much work in plan-

ning and plan recognition, the goal is to learn new (meta-level) control knowledge, not (object-level) domain knowledge.

In plan recognition, a sequence of operators is sought that deductively proves that the associated sequence of actions will achieve a goal. In ODYSSEUS, a sequence of operators (i.e., metarules) is also sought. In planning, complete information is usually available concerning the objects in the world, and this makes it easy to determine whether a given operator is applicable in the current world state. The world state information used by ODYSSEUS is based on beliefs concerning conclusions about object-level variables, so there is a great deal of plausible reasoning involved, as well as making and retracting beliefs in an attempt to find an explanation for the observed human behavior. Most plan recognition approaches to expert modeling, such as the MACSYMA-ADVISOR, HACKER, and BELIEVER, use a plan bug library approach (Genesereth, 1982; Sussman, 1976; Schmidt et al., 1978). The ODYSSEUS approach does not require a bug library of any sort.

### **2.2.6 Automatic Programming**

In automatic programming, the synthesis of LISP and PROLOG functions from example traces falls under the rubric of apprenticeship learning (Biermann, 1978; Shapiro, 1983a). The training examples consist of the input/output behavior of a correct program. The learning program modifies the performance element whenever the program being synthesized does not give the same output when given the same input.

## CHAPTER 3

### DETECTING DEFICIENCIES IN A KNOWLEDGE BASE

In the performance model of learning (Buchanan and Mitchell, 1978), detecting that the performance element has a deficiency is a prerequisite to making repairs to the object-level knowledge base. The detection of a deficiency requires comparison of the performance element to a *performance standard*. The performance standard used by different learning programs varies widely; it can be a metric that relates directly to the knowledge base or one that relates to a performance element that uses the knowledge base.

When learning by watching an expert, the performance standard used by ODYSSEUS to detect deficiencies is the expert's problem solving steps. When learning by watching a student, the performance standard is the knowledge base of the expert system. When learning from experience the performance standard is that the premises of strategy metarules should succeed. The main purpose of this chapter is describing the techniques that ODYSSEUS uses to detect deficiencies using these performance standards; these techniques are described in Sections 3.2 and 3.3. In order to provide perspective, Section 3.1 surveys the six most common performance standards used by machine learning programs to detect deficiencies.

<i>Learning performance standard</i>	<i>Learning program examples</i>
External: problem solution	META-DENDRAL, ARCH, INDUCE, RL, SEEK2, GENESIS, SIERRA
External: problem solution steps	Samuel, ARMS, ODYSSEUS
External: environment	HACKER, Genetic Algorithm
External: oracle — human	TEIRESIAS, LEAP
Internal: efficiency	LEX, LP, SOAR, Korf
Internal: correctness	AM, MORE, GUIDON-DEBUG

Table 3.1: All learning programs need a *performance standard* against which to judge the performance element to be improved. The six different standards used by existing learning programs are shown, along with examples of programs in each category. The standard used in a learning by watching an expert apprenticeship is the observed problem solution steps of a human expert.

### 3.1 Methods of Detecting Deficiencies

The six major ways of recognizing that a deficiency exists are shown in Table 3.1. The process of detecting deficiencies strongly localizes the source of the problem. This is especially true when the detection of deficiencies is accomplished by the use of an oracle.

#### 3.1.1 Problem Solution as External Standard

Learning programs are often given the correct solution to the problem being solved by the performance element. This correct solution serves as a performance standard. It is compared to the solution produced by the performance element to determine if the performance element is in error. For example, in trying to improve the DENDRAL performance element for determining chemical structure, META-DENDRAL is given mass spectrogram

data describing the chemical structure to be identified and the problem solution that is the correct chemical structure as determined by a human expert.

Winston's ARCH improves a performance element for classifying scenes as arches. ARCH is given a semantic network representation of a scene and the correct solution, i.e., whether the scene is an arch.

The ARCH program is a representative of the very large class of concept learning programs that are given a *classified* training instance, which consists of a problem and its correct solution. Learning programs for diagnostic expert systems, such as INDUCE, SEFK2, and RL, are also given a classified training instance.

Explanation based learning programs are also given a classified training instance, for example the GENESIS program. The program is given a kidnapping story and told that the story is a method of achieving wealth. Initially the performance element does not contain a schema for kidnapping. It can only explain a kidnapping story as a conjunction of two schemas. After being given the training instance, the GENESIS program is able to acquire the concept schema of kidnapping. The GENESIS program actually uses two of our external performance standards: the problem solution and the problem solution steps.

### 3.1.2 Problem Solution Steps as External Standard

A variation on being given the problem solution is to be given the individual steps that a problem solver might take to arrive at a correct solution. The classic example of this is Samuel's checker player program. Just being told the problem solution to a game of checkers would not be of much help to a learning program intent on improving a checker playing program. But by being given the substeps that lead to a winning game, Samuel's program can better localize the parts of the performance element that must be improved.

A recent very nice example of using problem-solving steps as the performance stan-

dard is the ARMS learning apprentice in the domain of robot assembly (Segre, 1979). Seeing the individual assembly steps allows ARMS to program a robot arm to mimic the observed assembly task. Note that unlike Samuel's checker play, the performance element's general ability is not increased, only its ability on problems that are identical in whole or part to those ARMS has previously observed.

The ODYSSEUS program described in this thesis shows that problem-solving steps also are a powerful aid to in learning to solve heuristic classification type problems. Previous learning programs in medical diagnosis, such as INDUCE, ID3, SEEK2, and RL, are given the set of data requests made by a physician as an unordered set, and also the final diagnosis. By contrast, ODYSSEUS treats each individual data request as a training instance.

### 3.1.3 Environmental Feedback as External Standard

Some learning programs use feedback from the environment of the performance element to detect a deficiency in the performance element. The performance element to be improved operates in a real-world or micro-world environment and the learning program has access to information on whether the performance element succeeds or fails at achieving a goal or subgoal. This requires that the goal of the performance element be explicitly stated, and that there is some means of determining whether the goal is met.

The genetic algorithm works in situations where there is a population of performance elements that are trying to achieve some goal (Holland, 1975; Holland, 1986). The performance elements with the better performance are saved and the other elements are discarded. These saved elements are modified in small ways, by operations such as point mutation or crossover. Crossover is a process of creating a new performance element by combining parts of two other performance elements. HACKER develops a plan to accom-

plish a goal, and monitors the execution of this plan. This allows detecting planning steps that fail to achieve a goal (Sussman, 1976).

#### **3.1.4 Oracle as External Standard**

An oracle can provide a performance standard. The oracle observes and critiques the performance element to be improved. This is the method used by TEIRESIAS. A physician watches MYCIN solve problems, and tells TEIRESIAS when a problem-solving action is wrong. The physician would also provide critiquing information as TEIRESIAS unwound the reasoning chain that lead to the inexplicable action. The LEAP apprentice in the domain of VLSI circuit design also relies on an oracle. When a circuit designer finds the VEXED circuit design aid to lack a needed circuit implementation rule, the circuit designer signals LEAP the the VEXED knowledge base is lacking a circuit implementation rule.

#### **3.1.5 Efficiency as Internal Standard**

It is possible for a learning program to have an internal standard of performance, and improve a performance element even though it never actually fails at a task. If solving a problem can be reduced to searching a problem space, learning from experience reduces to determining search knowledge that will to make the program more efficient. Often learning programs will analyze a performance trace of a performance element to determine if the performance element can be made more efficient. LEX and SOAR do this type of learning (Mitchell et al., 1983; Laird et al., 1987). Generally, programs that learn control knowledge would fall into this category.

### 3.1.6 Correctness as Internal Standard

The only two internal standards for detecting deficiencies are the efficiency and the correctness of the performance element.

A learning program might not know the correct solution but have a set of constraints that the performance element's knowledge structures or solution must meet. MORE and AM provide examples of constraints on knowledge structures. In the domain of diagnosis, MORE expects every finding to relate to at least one hypothesis. AM wished to see the slots of its mathematical concepts filled; an empty slot means that a concept is not fully defined, and this sent AM off to try to fill the slot. On heuristic classification problems, GUIDON-DEBUG requires that the solution explain all 'significant' findings. The "significant" findings constraint is also used by INTERNIST and PIP to determine when the final solution is adequate. Until this constraint is met, INTERNIST continues to gather more information on the patient.

## 3.2 Detecting Deficiencies When Learning by Watching

We will now provide the details of the technique for detecting discrepancies used by the two major apprenticeship scenarios: learning by watching and learning from experience. As the reader will recall from Chapter 1, the situation in which ODYSSEUS detects deficiencies when learning by watching is as follows. The human specialist is given a problem to solve and produces the type of behavior that was illustrated in Figure 1.5. The specialist may or may not provide goal information with each data request that is made. The objective of ODYSSEUS is to generate a set of explanations for each observed action of the specialist. When there are no explanations, ODYSSEUS assumes the performance element knowledge base has a deficiency.

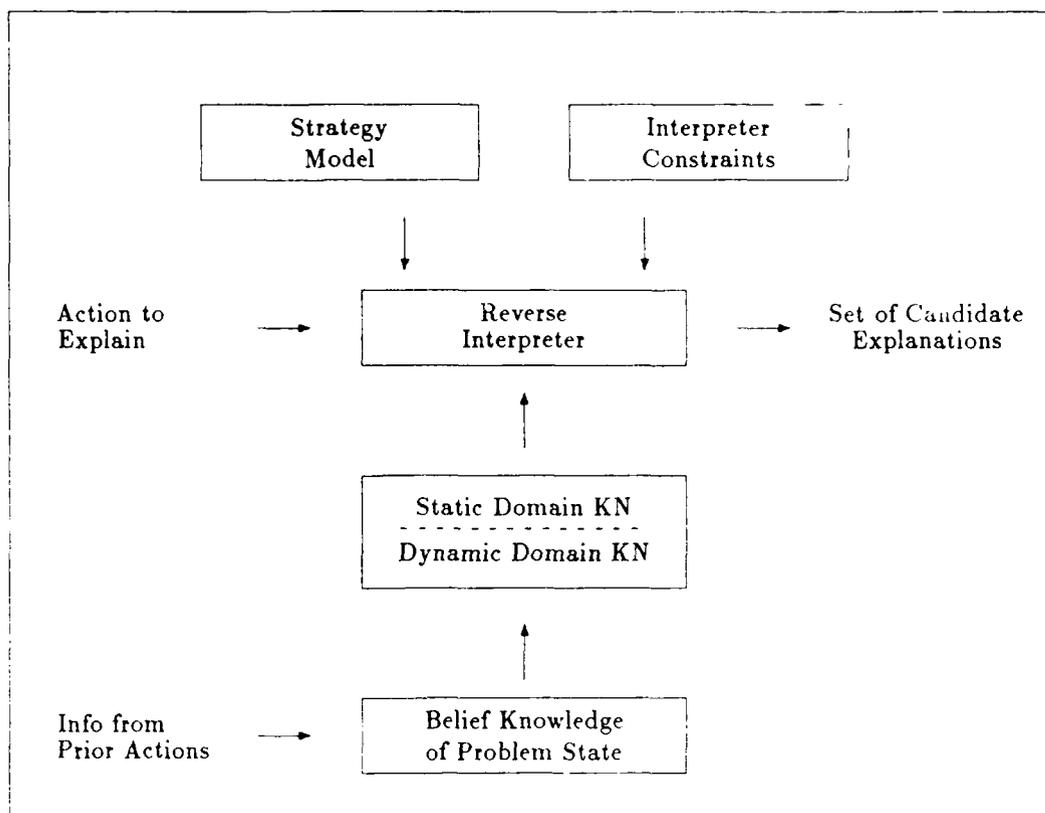


Figure 3.1: ODYSSEUS method of detecting discrepancies in a 'learning by watching' apprenticeship situation. An empty explanation set suggests a knowledge discrepancy.

### 3.2.1 Finding Explanations by Reverse Interpretation

For each observed action of the specialist, ODYSSEUS does a constrained generation of a set of Line of Reasoning Explanations (LOREs). A LORE relates an action  $A$  to an abstract strategic goal  $G$  via a skeletal rule path, that is,  $A \xrightarrow{R_1} T \xrightarrow{R_2} T \xrightarrow{R_3} \dots \xrightarrow{R_n} G$ . In our representation of LOREs,  $A$  is the action of an expert,  $R_i$  is a HERACLES metarule,  $T$  is a HERACLES' task, and the goal  $G$  is a HERACLES' task that represents a high-level general goal. Examples of high-level goals are *Test.Hypothesis*, *Group.and.Differentiate.Hypotheses*, and *Clarify.Finding*. Given  $A$ , all sequences of strategic metarules beginning with  $A$  and leading to a goal are in the LORE set of  $A$ ; from the perspective of the apprentice, the set delimits the possible interpretations that can be

attributed to the specialist's action *A*, in all possible worlds defined by the vocabulary of the expert system.

ODYSSEUS method of detecting discrepancies by the construction of LOREs can best be viewed as running the HERACLES expert system backwards. ODYSSEUS acts as a 'reverse interpreter' of HERACLES's metarules. For each observed action of the specialist, ODYSSEUS starts with the metarule *Request.data*, since this is the place where HERACLES performs observable actions, and backchains through the metarules in an attempt to create a path between the *Request.data* metarule and a high-level active goal.

HERACLES solves problems by starting with the high-level goal *Make.Diagnosis*, and problem solving at the meta-level involves forward chaining through metarules. Whenever a particular metarule within the task *Findout* is invoked, HERACLES performs an observable action, namely, it makes a data request. The HERACLES task interpreter is always trying to find a set of bindings for the clauses in a metarule premise; if it succeeds, then this metarule succeeds, and these bindings are passed to the action of the metarule.

By contrast, ODYSSEUS starts with a binding for a variable in the action of the *Request.data* metarule; the variable is bound to the value of the data request made by the expert. ODYSSEUS then tries to find a bindings for variables in the premise of the metarule that are consistent with earlier bindings. HERACLES uses a logic programming format for metarules, namely that of the logic programming language MRS, but actually translates the metarules into LISP code before HERACLES is run for efficiency reasons. This makes the reverse interpretation process a very difficult one. Until HERACLES can be run with a logic programming interpreter, ODYSSEUS must contain specialized code for each metarule that runs the metarules backward. One terminating condition used by the reverse interpreter of ODYSSEUS is only to search chains of metarules that are less than eight metarules long. Another constraint is to terminate the search for explanations

after the first one is found. This is allowable, since a discrepancy is detected only when the explanation set for a data request is null.

An important requirement in the reverse interpretation process is to know the dynamic state information of the expert system. This is obtained by ODYSSEUS by running HERACLES task Forward Reasoning, on each new piece of information on the patient that is learned.

### 3.2.2 The Strategy Space of Heracles

When ODYSSEUS runs HERACLES in a 'reverse interpretation' mode, it can be viewed as finding all interpretations that correspond to deletion and reordering of metarules, that is to the three inner rings shown in 3.2. The tasks for the metarules impose a rigid format on the order in which metarules are tried. The space of all reorderings of metarules represents a relaxation of this ordering constraint. When no LORE can be discovered this means that there is a domain knowledge difference or there is a strategic method used by the user that is not encoded in HERACLES's metarule. An example of a strategy often used by physicians that is not in HERACLES's metarules is a 'rule out' strategy. Such a strategy confirms the presence of a disease by ruling out all its logical competitors. HERACLES is currently not designed to use rules with a negative certainty factor, and so implementing a rule out metarule would not be easy.

A potentially more powerful learning method that might detect discrepancies missed by ODYSSEUS would be to generate only explanations (i.e., find metarule sequences) that are equivalent to HERACLES, which are a subset of the ODYSSEUS set, as Figure 3.2 makes clear. Behavior is equivalent to HERACLES if the differences are caused by allowable re-ordering of domain knowledge elements or metarule invocations. Such a modeler would be more powerful than ODYSSEUS, because in addition to detecting domain knowledge errors,

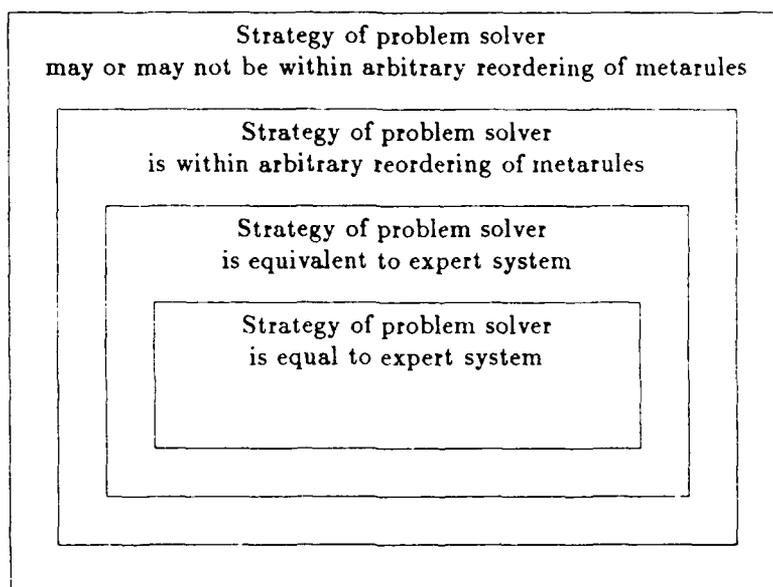


Figure 3.2: Degrees of strategy differences between specialist and expert system for a particular data request. ODYSSEUS' method finds all possible explanations assuming that the strategy of the expert is within an arbitrary deletion or reordering of strategy metarules.

it would detect situations where the difference between the program and the specialist is that the specialist is using a strategy that HERACLES finds inappropriate. This would be of particular interest when the specialist was a student. This is a much harder problem, and is actually the approach that an early student modeler for HERACLES, called IMAGE, attempted to take (London and Clancey, 1982). This approach requires running HERACLES through all forward moves, and is complicated by the fact that HERACLES liberally stores state information on property lists, on global variables, and on the LISP interpreter stack, so the expert system cannot easily be returned to an earlier problem-solving state. Doing a state-space search without complete state information is difficult. This approach will be more feasible when a Truth Maintenance System is added to HERACLES.

### 3.3 Detecting Deficiencies When Learning from Experience

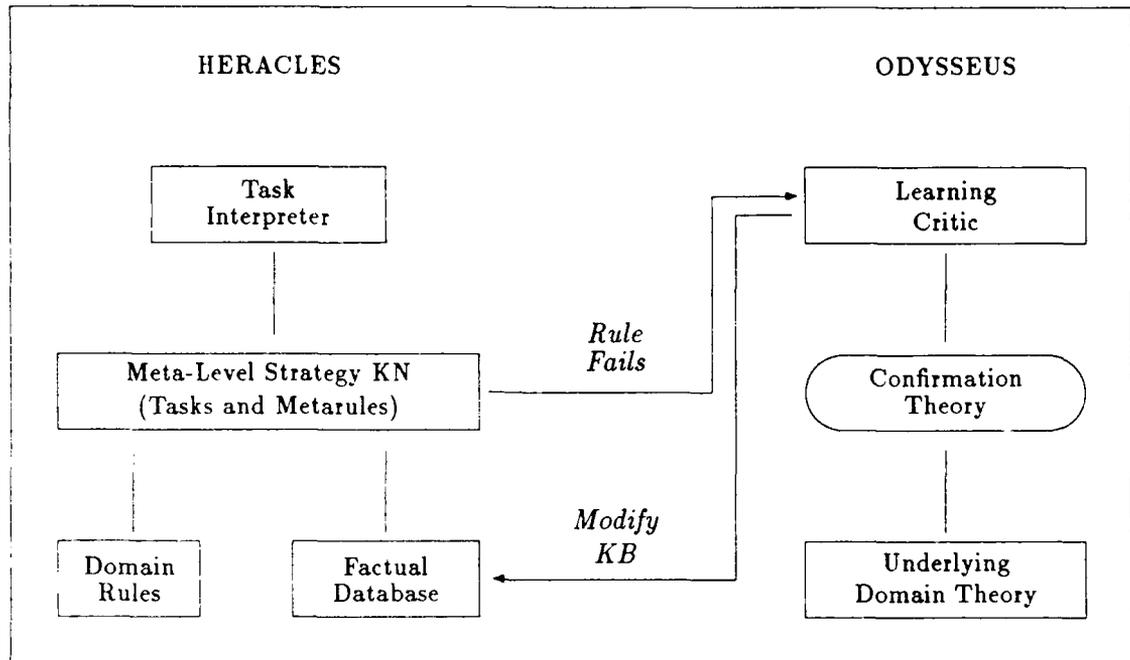


Figure 3.3: The relation between HERACLES and ODYSSEUS in a learning from experience apprenticeship scenario. If a metarule premise fails while HERACLES is solving problems, this suggests that there may be a gap in the knowledge base. ODYSSEUS attempts to locate knowledge that allows the metarule to succeed. If such knowledge is found, ODYSSEUS adds it to the HERACLES knowledge base.

A knowledge base gap will almost always cause a metarule premise failure. This section describes how control knowledge is monitored for failure of strategy metarule premises, thereby allowing the detection of gaps in the knowledge base. An overview of the learning method to be described is shown in Figures 3.3 and 3.4. This section also illustrates the role of strategy knowledge in the use of an underlying domain theory for learning. The strategy metarules of HERACLES reference object-level knowledge where different 'types' of object-level knowledge, in a programming language sense, are represented in different EDB relations. This provides a foundation for the integration of an

underlying domain theory into a learning system, because justification of different types of new knowledge usually requires different ways of using an underlying domain theory. We advocate the construction of a decision procedure for each EDB relation that specifies how the relation is defined and justified in terms of an underlying domain theory.

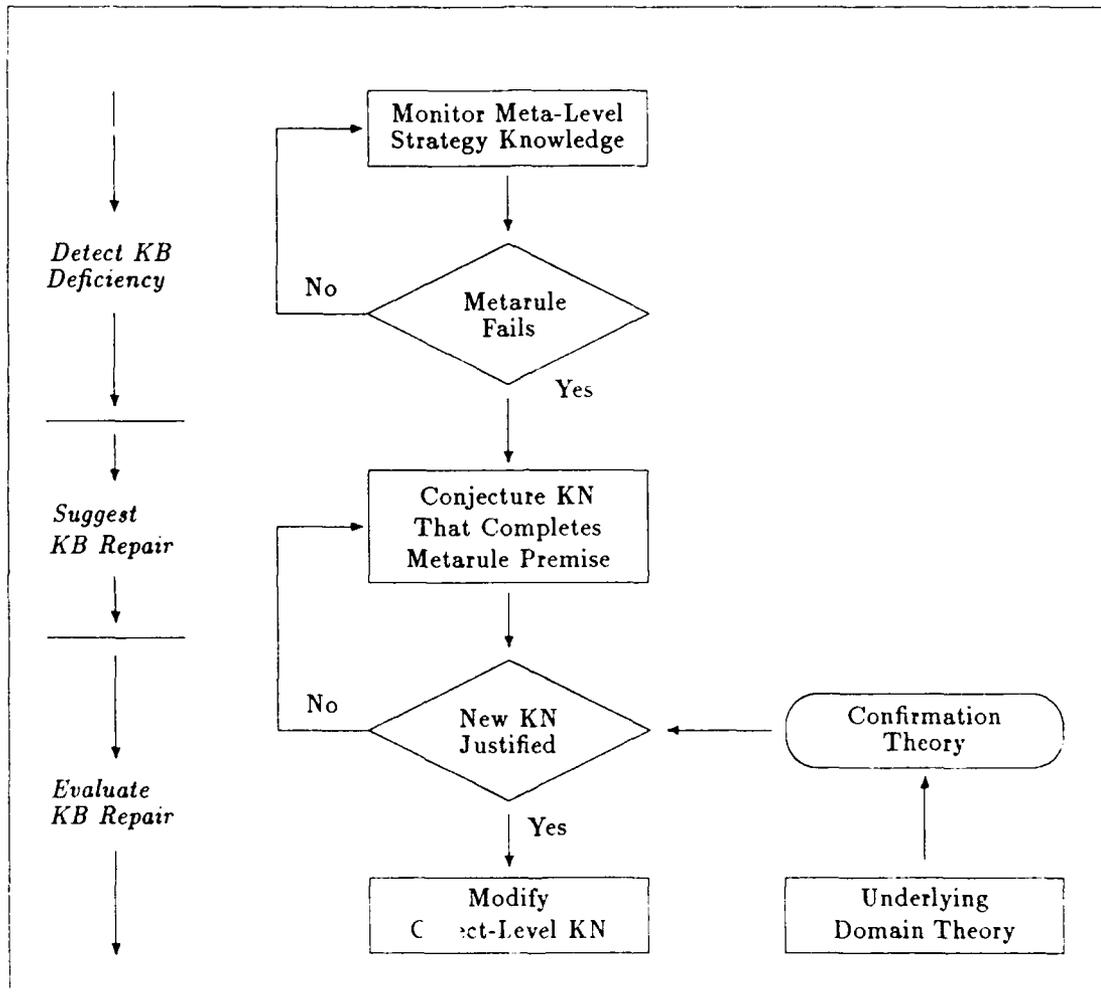


Figure 3.4: ODYSSEUS' method for *learning from experience*. ODYSSEUS monitors the problem solving behavior of an expert system. Failure of a metarule suggests a knowledge base discrepancy. The second and third stage are very similar to the learning by watching scenario.

A deficiency in the knowledge base is suspected whenever the premise of a metarule fails. The preconditions of HERACLES' metarules fail between 60% and 70% of the time. In a typical NEOMYCIN consultation 500 to 2000 metarules premises are tried. The

overriding reason for failure is that the knowledge base does not contain the knowledge necessary to instantiate the EDB relations in the metarule premise. These are potential learning opportunities. ODYSSEUS monitors metarule failures and then determines which relations in the premises are responsible for the failure. If a failed relation indexes dynamic state information or is used to control the meta-level reasoning, then there is no object-level learning opportunity. However, if a failed conjunct is the type that accesses the domain knowledge base, then this is a learning opportunity. Given this failed clause, the second and third stages of ODYSSEUS attempt to suggest and evaluate changes to the knowledge base related to the clause that would allow the failed clause to succeed. If such knowledge is found and is added to the knowledge base, the metarule will succeed in the future in a similar situation.

## CHAPTER 4

### SUGGESTING REPAIRS TO A KNOWLEDGE BASE

Given evidence of a knowledge base deficiency, a learning program must suggest (or request an expert to suggest) a repair to the knowledge base that addresses this deficiency. Methods of suggesting repairs take as their starting point localization information provided by the method that detects deficiencies. For example, when ODYSSEUS cannot explain an action and thereby detects a potential deficiency, the suggested repair is directly or indirectly related to the action.

Extant learning programs use a variety of methods of suggesting these repairs. In this chapter, we first survey the seven most common techniques used by learning programs. The techniques used by ODYSSEUS are then presented.

#### **4.1 Techniques for Suggesting Repairs**

The seven most commonly used techniques for suggesting repairs, along with examples of learning programs for each of these techniques, is shown in Table 4.1. Each of these will be discussed in turn.

### 4.1.1 Mutation and Crossover

Giving a population of performance elements, a novel means of suggesting repairs is to combine or 'crossover' pieces of successful performance elements. Crossover is a form of survival of the fittest and is much more sophisticated than random mutation. Holland's genetic algorithm pioneered this approach (Holland, 1975).

AM mutated LISP code to learn high-level symbolic structures (Lenat, 1976). This approach worked because of the close similarity between LISP and the high-level symbolic concepts to be learned, which were concepts from elementary set theory. The usefulness of using this approach to learning high-level symbolic concept structures in other domains by mutation has not yet been successfully demonstrated.

<i>Technique that suggests KB repair</i>	<i>Learning program examples</i>
Mutation, Crossover	Genetic Algorithm, AM
Weight Adjustment	Samuel, SEEK2, Connectionist
Rule Generalization	META-DENDRAL, INDUCE, RL, LEX, LEAP
Plan Generalization	SOAR, BUCKET-BRIGADE, ARMS, GENESIS
Plan Completion	SIERRA, PRE, ODYSSEUS
Analogy	MACBETH, NLAG, Russell
Oracle — human	TEIRESIAS, LEAP, MORE, GUIDON- DEBUG

Table 4.1: The different methods used by learning programs to suggest knowledge base repairs.

### **4.1.2 Weight Adjustment**

The best examples of weight adjustment are the connectionist approach to AI and the version of Samuel's checker player that used a polynomial evaluation function (Samuel, 1963). Learning is reduced to the adjustment of weights.

### **4.1.3 Rule Generalization**

A common technique for suggesting changes is by generalizing or specializing rules. When the performance element produces the incorrect answer, we changes consistent with past solved cases that allow the performance element to work better, which is a form of generalization grounded in similarity based learning. Alternatively, we can take an explanation based learning approach to the generalization or specialization. The various ways a knowledge element can be generalized or specialized is very nicely described by Dietterich and Michalski (Dietterich and Buchanan, 1981).

### **4.1.4 Plan Generalization**

Generalizing a plan (by generalizing an operator that is part of the plan) is another way of postulating a change. Backward propagation and goal regression are popular means of locating the part of the plan that requires generalization or specialization. Most EBG/EBL systems use this approach as does SOAR (DeJong, 1986; Mitchell et al., 1986; Laird et al., 1984)

### **4.1.5 Plan Completion**

In plan completion, the sequence of operators that form part of the plan are not changed. However, knowledge that binds to the operators is modified.

SIERRA examines all minimal modifications to a procedure for subtraction, in

search of one that provides an explanation of observed student subtraction bugs (VanLehn, 1983). In ODYSSEUS, the clauses of a LORE specify constraints that a declarative knowledge base must satisfy for the LORE to be true. ODYSSEUS can be viewed as searching the space of minimal knowledge base modifications in search of one that will satisfy all the constraints imposed by a LORE.

#### **4.1.6 Analogy**

Analogy is an example of a deductive method for suggesting knowledge base repairs. Determinations fall into this category (Russell, 1986).

#### **4.1.7 Oracle**

A cooperating human expert provides a learning program with an oracle for suggesting repairs. Manual methods of postulating changes to a knowledge base include MORE, GUIDON-DEBUG, TEIRESIAS, and LEAP (Kahn et al., 1985; Clancey, 1987; Davis, 1982; Mitchell et al., 1985). In the LEAP system, after the circuit designer signals LEAP that there is a missing implementation rule, the circuit designer must enter the implementation rule into the computer. LEAP generalizes this rule, if this is required. In a similar fashion, TEIRESIAS requires that a cooperating expert enter the missing rule or modify an existing rule. Given this information, TEIRESIAS partially validates the correctness of the entered rule.

## **4.2 Suggesting Repairs When Learning by Watching**

We will now provide a description of the technique used by ODYSSEUS to suggest object-level repairs when learning by watching. Recall that a knowledge base deficiency is detected when a human problem solver performs an action, and ODYSSEUS is unable to

create a LORE explanation that shows how the action relates to a high-level goal. The subsystem of ODYSSEUS that detects deficiencies provides the subsystem of ODYSSEUS that suggests repairs with three important pieces of information: the name of the unexplained action, an ordered list of the most likely high-level goals to which this action relates, and beliefs regarding dynamic state information such as what hypotheses and data requests are believed to be false. Determination of the beliefs of the human problem solver is made by running the expert system on all past data requests.

The method used to suggest repairs is to relax the constraints on the construction of LOREs. We begin with the single fault assumption, allowing a metarule to become part of a LORE, even though a clause in the metarule premise fails. If allowing a single clause in a single metarule to fail results in the creation of a LORE, then any EDB tuple that allows the clause to unify is a suggested repair.

### **4.3 Suggesting Repairs When Learning From Experience**

We now describe the technique used by ODYSSEUS to suggest knowledge base repairs when learning from experience. Only part of the value of an apprenticeship derives from having the novice watch experts solve problems; much of the value derives from having the novice attempt to use his or her fledgling expertise to solve problems. Our method concentrates on learning by doing, where learning occurs when the problem solver lacks the domain-specific knowledge to apply problem-solving methods.

Suggesting repairs when learning by experience is very similar to the situation of learning by watching. Indeed, it uses the same code, and provides much more focused input than in learning by watching.

In learning from experience, the subsystem of ODYSSEUS that detects discrepancies provides the subsystem of ODYSSEUS that suggests repairs with very detailed information.

The input to the subsystem that suggests repairs is the name of the metarule that failed, the known bindings for variables in the clauses of the metarule premise that have been determined outside of the scope of the metarule, and a knowledge of the range of values that each variable in a metarule clause is allowed to assume. So notice we are in the same situation as learning by watching, but do not have to locate the metarule that failed, nor require the deficiency detector to conjecture the high-level goal. These are the advantages to being able to tap directly into problem solver, which is possible when the problem solver is an expert system. As in learning by watching, the repair suggester generates all allowable variable bindings for the unbound variables in the failed clause. These tuples are then passed on to the ODYSSEUS subsystem that evaluates proposed repairs.

#### 4.4 Example of Suggesting Repairs

An example will help to make clear the process of suggesting repairs for the learning by watching scenario. The output of the subsystem that detects deficiencies is the name of the unexplained action, which in this case is visual problems, and the potential focus of the high-level goal, which is viral meningitis. The ODYSSEUS subsystem that generates repairs produces the following conjectures:

```
evidencefor(diplopia acute-meningitis R1 cf)
evidencefor(photophobia acute-meningitis R1 cf)
general-question(visual-problems)
clarify-questions(nausea visual-problems)
```

The first candidate repair suggests that there is a rule missing that contains diplopia in its premise and provides evidence for acute meningitis in its conclusion. The second candidate is similar, and suggests a rule is missing that contains photophobia in its premise

and provides evidence for acute meningitis in its conclusion. The third candidate repair suggests that visual problems is a general question that should be asked of all patients. The fourth repair states that visual problems is a clarifying question for nausea. Inclusion of any of these repairs in the knowledge base would lead to a completed explanation of the physician's action. The candidates are generated in order of decreasing plausibility. Normally, ODYSSEUS evaluates each candidate as it is generated, and stops when the first candidate is accepted by the repair evaluator. Evaluation of repairs is the subject of the next chapter.

We will give a detailed explanation of how the first two candidate repairs are generated. By allowing a single clause in a single premise to fail, many metarule chains are successfully constructed between the unexplained action and a high-level goal. The chain associated with the first two candidate repairs shown above consists of six metarules; they form a chain from the action visual problems to the high-level goal group.and.differentiate.hypothesis viral.meningitis. The goal in the action of one metarule connects to the same goal in the premise of the next metarule. The six metarules are as follows.

**MetaRule 1: Group.and.Differentiate.Hypotheses**

```

IF:      goal(group.and.differentiate.hypotheses($hypothesis1)) ^
         active.hypothesis($hypothesis1) ^
         parent($hypothesis1 $hypothesis2)
THEN:    goal(test.hypotheses($hypothesis2))

```

ENGLISH: If the current goal is to group and differentiate hypotheses  
and there exists an active hypothesis  
and this active hypothesis has a parent  
and the likelihood of this parent is currently unknown

then test this parent hypothesis.

**MetaRule 2: Test.Hypothesis**

IF:       goal(test.hypothesis(\$hypothesis2)) ^  
           evidence.for(\$finding1 \$hypothesis2 \$rule1 \$cf1) ^  
           unknown(rule.applied(\$rule1))

THEN:     goal(applyrule(\$rule1))

ENGLISH: If the current goal is to test a hypothesis  
           and there is a rule that gives evidence for this hypothesis  
           and this rule is unknown  
           then apply this rule.

**MetaRule 3: Applyrule**

IF:       goal(applyrule(\$rule1)) ^  
           in.premise(\$finding1) ^  
           unknown(parm.applied(\$finding1))

THEN:     goal(findout(\$finding1))

ENGLISH: If the current goal is to apply a rule  
           and a finding in the premise of the rule is unknown  
           then try to find out the value of this finding.

**MetaRule 4: Findout.by.Subsumption**

IF:       goal(findout(\$finding1)) ^  
           subsumed.by(\$finding1 \$finding2) ^

```

boolean($finding2) ^
unknown(concluded($finding2)) ^
askfirst($finding2)
THEN: goal(findout $finding2)

```

ENGLISH: If the current goal is to find out a finding  
 this finding is subsumed by another finding  
 and this parent takes boolean values  
 and this parent is currently unknown  
 and to find out the value of this finding first ask the user  
 then find out the value of this parent finding.

#### MetaRule 5: Findout.by.Asking.User

```

IF: goal(findout($finding2)) ^
askfirst($finding1 $finding2) ^
not(value-known($finding2))
THEN: goal(ask-user($finding2))

```

ENGLISH: If the current goal is to findout the value of a finding  
 and to findout the value of this finding first ask the user  
 and finding2 is currently unknown  
 then ask the user the value of this finding.

The previous metarule chain can be collapsed into the following rule. This rule shows all the clauses of domain knowledge that are relevant to explaining why the action of asking visual problems relates to achievement of the high-level goal

group.and.differentiate.hypothesis viral meningitis.

```

IF:      goal(group.and.differentiate.hypotheses($hypothesis1)) ^
        active.hypothesis($hypothesis1) ^
        parent($hypothesis1 $hypothesis2) ^
        evidence.for($finding1 $hypothesis2 $rule1 $cf1) ^
        unknown(rule.applied($rule1)) ^
        in.premise($finding1) ^
        unknown(parm.applied($finding1)) ^
        subsumed.by($finding1 $finding2) ^
        boolean($finding2) ^
        unknown(concluded($finding2)) ^
        askfirst($finding2) ^
        askfirst($finding1 $finding2) ^
        not(value-known($finding2))

THEN:    goal(ask-user($finding2))

ENGLISH: If goal is to group $hyp1
        then ask the user the value of this finding.

```

In the example in which our learning critic was called into play, \$active.hypothesis consisted of seven hypotheses, including viral meningitis. Metarule 2 fails because a binding for \$finding cannot be found in the relation evidence.for. Other clauses establish bindings for \$hypothesis1 and \$hypothesis2 in this metarule. Using information regarding the range of permissible values for \$finding, the learning critic conjectures the two bindings for evidence.for shown at the beginning of the example.

## CHAPTER 5

### EVALUATING REPAIRS TO A KNOWLEDGE BASE

The last of the three stages of apprenticeship learning is evaluation of suggested knowledge base repairs. The method we propose for apprenticeship learning is radically different from previous evaluating approaches. It involves the use of a confirmation theory that specifies how to determine the value or desirability of an arbitrary candidate of domain knowledge, i.e., an arbitrary tuple. The confirmation theory justifies EDB tuples using several knowledge sources, including an underlying theory of the domain.

<i>Validation standard</i>	<i>Learning program examples</i>
Accuracy	META-DENDRAL, INDUCE, SEEK2
Efficiency or accuracy	Samuel, Genetic Alg., LEX, RL
Oracle — human	TEIRESIAS, MORE, GUIDON-DEBUG, AM
Underlying domain theory	LEAP, ODYSSEUS

Table 5.1: Methods of evaluating knowledge base repairs used by learning programs.

## 5.1 Spectrum of Techniques for Evaluating Repairs

The number of ways to evaluate a proposed knowledge base repair is not that large; the commonly used methods are summarized in Table 5. For algorithmic and deductive learning methods, such as version spaces, no validation is required: the learning method is guaranteed to give the correct solution.

The most common means used by other researchers is to measure the improvement to the performance element, which accords well with those who believe in the performance model of learning shown in Figure 1.3.

A less exciting but perfectly fine method is to ask an expert to evaluate the proposed repair.

Finally, there is the method of using an underlying domain theory. That is, given a candidate piece of knowledge, an underlying domain theory can be used to verify whether the knowledge is true or false. This is not the same as using domain theories as 'half-order' theories to guide the search for knowledge base improvements, as was done in META-DENDRAL and SIERRA (Lindsay et al., 1980; VanLehn, 1983).

## 5.2 The Confirmation Theory

The confirmation theory provides a decision procedure for each relation in the knowledge base. Given an arbitrary tuple, it decides whether adding that tuple to the knowledge base would improve the knowledge base. An example of an input to the confirmation theory is the tuple `subsumes(visual-problems, double-vision)`. Our decision procedures may refer to the existing knowledge base in forming their evaluation of a tuple. As an underlying domain theory, our decision procedures use an induction system that operates over a library of past solved cases, along with an explanation generator that

operates over HERACLES strategy knowledge of tasks and metarules.

### 5.2.1 Knowledge- vs. Performance-oriented Validation

The ultimate goal of a learning program is to improve the performance of a problem solver or performance element. However, the architecture of knowledge based systems requires a shift in our concept of improved performance. We refer to the type of validation technique we advocate as *knowledge-oriented* validation and distinguish it from the traditional practice of *performance-oriented* validation.

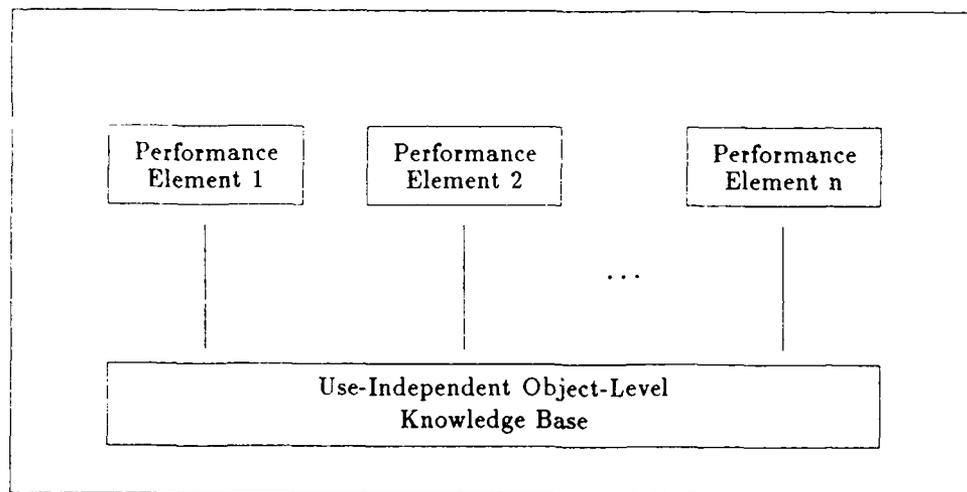


Figure 5.1: A use-independent knowledge base provides object-level knowledge to multiple performance elements, and this complicates evaluation of knowledge base repairs. Our solution approach involves the use of a confirmation theory for evaluating repairs.

Performance-oriented validation requires that modifications to a particular problem-solving program improve problem-solving performance. Because this type of validation has traditionally focused on improved performance with respect to a single problem-solving program, the veracity of the underlying knowledge has not been of overriding concern. A system designed exclusively to maximize problem-solving performance of a particular problem-solving program may use a method of knowledge representation in which the

semantics of the domain knowledge cannot be represented easily, if at all. A polynomial evaluation function for rating checker positions, for example, does not capture all the meaning of its terms.

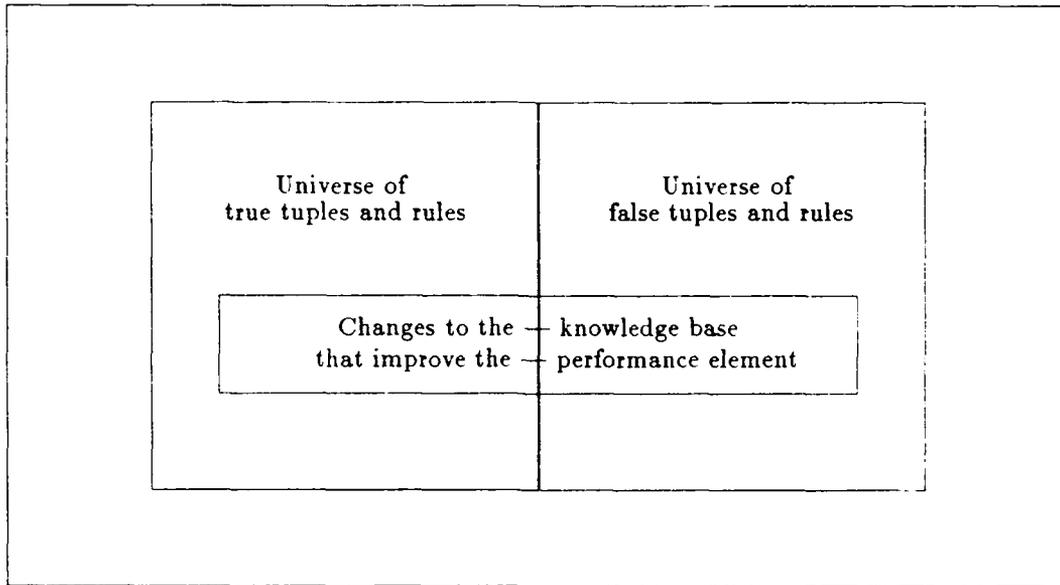


Figure 5.2: With a use-independent knowledge base, knowledge should not be added to a knowledge base just because it is true, nor just because it improves performance. Ideally, we would like both these conditions to be true.

Knowledge-oriented validation might be defined as performance-oriented validation that prohibits lessening the quality of individual pieces of knowledge elements solely for the sake of problem-solving performance. The advent of large declarative knowledge bases used by multiple problem solving programs makes this perspective important. Examples of multiple problem-solving programs that might use the same medical knowledge base are programs to accomplish medical diagnosis, knowledge acquisition, intelligent tutoring, and explanation. When multiple programs use the same declaratively-specified knowledge base, it is helpful to specify knowledge in a manner that is independent, so far as possible, of its use. Knowledge-based validation accomplishes this by requiring that changes to the knowledge base be semantically meaningful.

Suppose we wish to be faithful to the traditional performance-oriented validation paradigm when using multiple-purpose knowledge bases. This requires that every time a learning program finds a change to the knowledge base that will improve one problem solving program, before that change can be recorded, the validation method must insure that the aggregate performance of all programs is improved. This policy will be expensive and computationally overwhelming. Further, programs for all the intended uses of a knowledge base are not necessarily in existence at the time learning is taking place.

Another rationale for knowledge-oriented validation is our belief that performance in the long term will be more correct and robust if the knowledge structures are carefully developed. Moreover, when the problem solver is a human, it is unrealistic, probably even unwise, to attempt to replace semantically-rich knowledge structures with others that deviate radically from them merely to improve short-term performance.

It should be noted that to some extent all programs for improving an intelligent agent aim at both good performance and good knowledge; nevertheless, almost all past research in machine learning, intelligent tutoring and automatic programming has adopted a pure performance-oriented validation approach. This is especially true in automatic programming, where any mutation to the program to be debugged is judged to be acceptable if it causes the program to produce the correct output when given a correct input/output training instance (Shapiro, 1983a).

### **5.2.2 Examples of Using a Confirmation Theory**

In the remainder of this section, two learning examples will be described in detail to demonstrate the approach we are advocating. The first example, given in Section 5.2.3, illustrates the learning of tuples for the EDB relation clarifying questions, using the ODYSSEUS strategy theory as the underlying domain theory. The second example, given

in Section 5.2.4, illustrates the learning of rule knowledge for the relation `evidence.for`, using a confirmation theory based on induction over past cases as the underlying domain theory. These examples are based on the NEOMYCIN knowledge base, the MYCIN case library, and an actual medical case. Both sections assume that a metarule failure has occurred and that candidate repairs have been generated; they concentrate on the third stage of learning, wherein candidate repairs are tested.

### 5.2.3 Confirmation Theory for Factual Knowledge

The focus of this example is the `clarifying.questions` clause in the `clarifying.questions` metarule presented below. As an example of its use, suppose the doctor discovers that the patient has a headache. The headache finding is associated with many diagnostic hypotheses, so many that it is generally wise to narrow down this set of hypotheses by determining the severity and duration of the headache before pursuing a specific hypothesis. This is the process of *clarifying the finding*, and the questions about various subtypes of this finding (e.g., headache-duration, headache-severity) are called *clarifying questions*. In the HERACLES system, this is implemented by invoking the `clarify.finding` task whenever a new finding is derived by the system or provided by the user. In turn, the `clarify.finding` task invokes the `clarifying.questions` metarule.

#### MetaRule 1: Clarify.questions

```

IF:      goal(clarify.finding $finding1) ^
         clarifying.questions($finding1 $finding2) ^
         not(value-known $finding2)

THEN:   goal(findout $finding2)

```

ENGLISH: If the current goal is to clarify finding1  
 and finding1 can be clarified by finding2  
 and finding2 is currently unknown  
 then try to find out the value of finding2.

Only one of the premise conjuncts of Metarule 1 accesses domain knowledge, namely clarifying.questions(\$finding1 \$finding2). The first conjunct is for control purposes and the third conjunct checks the value of dynamic state knowledge.

Learning may occur when Metarule 1 is passed a value for the variable \$finding1, say headache, but Metarule 1's premise fails because no bindings can be found for \$finding2. In this situation, \$finding2 is a free variable at the time of failure. ODYSSEUS begins the learning process by invoking the candidate repair generator, which generates every plausible candidate binding for \$finding2. Using information regarding the range of \$finding2, the learning critic is able to generate about 300 candidate relations.

In order to be able to evaluate new domain knowledge, two steps must be taken beforehand. First, specify all the constraints that an instance of the EDB relation must satisfy in order to be valid. In our example, this requires constructing a precise definition that captures the constraints on an instance of the clarifying.questions relation. Second, test these constraints using an underlying theory of the domain. This two-step method contrasts with the current manual method of refining the NEOMYCIN knowledge base, which consists of directly asking physicians for new knowledge.

Let us begin by giving an informal justification of clarifying.questions. One reasonable justification for asking clarifying questions is cognitive economy with respect to efficient diagnosis. That is, clarifying questions can reduce the number of questions that a physician must ask in order to arrive at the correct diagnosis. Much of diagnosis involves the testing of specific hypotheses; however, sometimes a new piece of information is

discovered that suggests a very large number of hypotheses. To reduce the number of relevant hypotheses, it is helpful to ask several clarifying questions that will add confirming or disconfirming evidence to many of the hypotheses associated with the new piece of information. After asking these questions, only a few of the numerous potential hypotheses will now be consistent with what is known.

We can now give a precise description of the constraints operating on clarifying questions. An explicit description of a clarifying question is as follows: if a question (finding) is associated with many hypotheses, say more than six, and there exists a question that provides positive or negative evidence to many of these hypotheses, say between one-third and two-thirds, then always ask this question as a clarifying question. This can be formalized as follows.

**Definition 1.**

For any finding  $f$ , let  $H_f$  be the set of all hypotheses  $h$  such that  $\text{relatesTo}(f, h)$  is true. Let  $f_1$  and  $f_2$  be distinct findings, such that  $\text{subsumes}(f_1, f_2)$  is in the knowledge base. Let  $n$  be an empirically determined threshold indicating the minimum number of hypotheses that a finding must relate to in order to require the use of clarifying questions. Then

$$\text{clarifying.question}(f_1, f_2) \longleftrightarrow [(\|H_{f_1}\| \geq n) \wedge (\frac{1}{3}n \leq \|H_{f_1} \cap H_{f_2}\| \leq \frac{2}{3}n)].$$

◇

The  $\text{relatesTo}()$  relation is not part of the domain knowledge base; it is computed on the fly when a new piece of knowledge is evaluated, using the ODYSSEUS LORE generator described in detail in Chapter 3. The LORE generator can enumerate all the reasons that a question could possibly be asked, given the strategy and domain knowledge in HERACLES. The line of reasoning generator allows determination of all the hypotheses

that are associated with any one question either directly or indirectly; it is used to compute  $\text{relatesTo}(f, h)$ .

We now describe the results of encoding Definition 1 and implementing our approach for the NEOMYCIN knowledge base. Currently, there are two clarifying questions for headache in the NEOMYCIN knowledge base: headache duration and headache severity. ODYSSEUS considered the effect of all headache-related questions on the set of hypotheses associated with headaches, and determined that one more clarifying question met the above described constraints: headache progression (i.e., is the headache getting better or worse). ODYSSEUS automatically modified a slot value under headache in the knowledge base to include this clarifying question; in the future, this question will always be asked when the patient complains of a headache.

#### 5.2.4 Confirmation Theory for Rule Knowledge

All rule knowledge is represented within HERACLES using EDB and IDB relations. This means that rules can be learned much as factual knowledge is learned. The example in this section involves learning a tuple of the `evidence.for` relation through the `split.active.hypotheses` metarule. This rule is one of three invoked by the task `group.and.differentiate.hypotheses`. This metarule is useful during diagnosis when there are currently a large number of strong diagnostic hypotheses. The `split.active.hypotheses` metarule searches for a data request to ask that will simultaneously provide strong positive evidence for some active hypotheses and strong negative evidence against other active hypotheses.

##### MetaRule 2: Split.Active.Hypotheses

```
IF:      goal(group.and.diff.hypotheses $active.hypotheses) ^
         member($hypothesis1 $active.hypotheses) ^
         member($hypothesis2 $active.hypotheses) ^
```

```

not(equal($hypothesis1 $hypothesis2)) ^
evidence.for($finding $hypothesis1 $rule1 $cf1) ^
evidence.for($finding $hypothesis2 $rule2 $cf2) ^
greater($cf1 .2) ^
less($cf2 -.2)

```

THEN: goal(findout \$finding)

ENGLISH: If the current goal is to group and differentiate a list of active hypotheses and a single finding provides positive evidence for one of the hypotheses and negative evidence for another of the hypotheses then try to find out the value of this finding.

The metarule is passed a value for the variable \$active.hypotheses. The interpreter attempts to find a unifier for all the clauses such that \$hypothesis1 is bound to one member in \$active.hypotheses, \$hypothesis2 is bound to a different member of \$active.hypotheses, and there is a single finding in the premise of a metarule that concludes that \$hypothesis1 is probably present and is also in the premise of a rule that concludes that \$hypothesis2 is probably absent. That is, a finding is asked that simultaneously provides evidence against some of the hypotheses and evidence for other hypotheses. Even though the NEOMYCIN knowledge base has been under development for several years, the split.active.hypotheses metarule is rarely invoked on any of the patient cases in the NEOMYCIN case library.

In the example that our learning critic was called into play, \$active.hypotheses consisted of seven hypotheses: AV malformation, mycobacterium TB meningitis, viral meningitis, acute bacterial meningitis, brain aneurysm, partially treated bacterial menin-

gitis, fungal meningitis. The metarule fails because a binding for \$finding cannot be found in the two relations `positive.evidence.for` and `negative.evidence.for`. Other clauses establish bindings for \$hypothesis1 and \$hypothesis2. Using information regarding the domain of \$finding, ODYSSEUS conjectures many potential missing rules. The number of conjectures can be quite large.

Given these conjectures, a confirmation theory determines whether any of them is true. This requires the use of a decision procedure for the \$evidence.for relation based on an underlying domain theory.

**Definition 2.**

Let  $r$  be a justifiable domain rule. Let  $f$  be a finding that appears in the premise of  $r$ , and let  $h$  be a hypothesis that appears in the conclusion of  $r$ . Let  $s$  be the certainty factor strength of  $r$  that indicates the evidence of  $f$  for  $h$ , normalized to lie between +1 and -1. Let there exist predicates `general`, `specific`, `complex`, and `collinear`, that return true if the rule is, respectively, sufficiently general, sufficiently specific, above a complex threshold, and a collinear version of an existing rule in the knowledge base. Then

$$\text{evidence.for}(f, h, r, s) \longleftrightarrow$$

$$\text{general}(r) \wedge \text{specific}(r) \wedge \neg \text{complex}(r) \wedge \neg \text{collinear}(r).$$

◇

ODYSSEUS uses induction over a case library to determine whether the conjectured rule meets the constraints of the definition. A statistical analysis of the past solved cases determines whether a rule is sufficiently general and sufficiently specific. An analysis of the existing knowledge base determines whether a rule is a collinear version of an existing rule. The rule complexity is measured by the number of conjunctions the rule contains.

The confirmation theory using the ODYSSEUS induction system found five rules that divide the list of active hypotheses, including:

**Object-Level Rule 1.**

IF:       duration.of.symptoms  $\leq$  1 day  $\wedge$

          evidence.for(meningitis)  $\geq$  .6

THEN:   suggests fungal.meningitis (cf = -.8)  $\wedge$

          suggests mycobacterium.tb.meningitis (cf = -.8)  $\wedge$

          suggests acute.bacterial.meningitis (cf = .7)

## 5.3 The Underlying Domain Theory

### 5.3.1 Odysseus' Induction System

The induction subsystem of ODYSSEUS serves as an underlying basis for the 20% of the knowledge base that consists of heuristic associational rules. The induction subsystem of ODYSSEUS is principally concerned with searching the space of rules of the form *lhs*  $\rightarrow$  *hypothesis* (CF), where CF is a MYCIN-type certainty factor. A candidate rule evaluator checks the *lhs* of candidate rules to see whether they meet given constraints of minimal rule generality (coverage), minimal rule specificity (discrimination), maximal rule collinearity (similarity), and maximal rule simplicity (number of conjunctions and disjunctions). The rule evaluator always gives preference to collinear forms of heuristic rules contained in the original rule base.

Some EDB relations may not be justifiable using empirical methods that operate over a set of past solved problems.

### 5.3.2 Related Research

Two major apprenticeship learning systems are LEAP and DIPMETER ADVISOR (Mitchell et al., 1985; Smith et al., 1985). In both of these systems there is a single *type* of knowledge. In LEAP, all knowledge is implementation rules. In DIPMETER ADVISOR all knowledge is heuristic rules. In contrast, there are dozens of types of knowledge in HERACLES—each EDB relation corresponds to a different type of knowledge. The key to automatic learning seems to be the definition of constraints to tie each relation individually to an underlying domain theory. The DIPMETER ADVISOR uses explicit justifications that are tailored for each heuristic rule. By contrast a single confirmation theory is used by ODYSSEUS to provide an explicit justification for all heuristic rules. The DIPMETER ADVISOR provides an example of the use of very complex rule justifications for heuristic rules.

ODYSSEUS has a separate decision procedure for each EDB relation. This is reminiscent of the approach taken in in AM (Lenat, 1976), where each slot of a concept has a set of associated heuristic rules that can be used to evaluate the contents of the slot.

## 5.4 Example of Evaluating Repairs

Returning to the example that was described in Chapters 1 and 4, the candidate repair evidence.for(photophobia acute-meningitis \$rule \$cf) is submitted to the repair evaluation subsystem. The induction system finds a good rule the includes photophobia in the premise and acute-meningitis in the conclusion, as documented in the following transcript.

Second candidate selected.

A good rule corresponding to the evidencefor is:

IF patient-finding(photophobia yes)

THEN conclude(acute-meningitis .7)

Rule goodness measures:

Rule generality: .22

Rule specificity: .76

Rule collinearity: .06

Rule simplicity: 1

The explanation that this completes is as follows:

IF conclude(visual-problems no)

THEN conclude(photophobia no) by subsumption.

IF conclude(photophobia no)

THEN conclude(acute-meningitis no .7)

via a heuristic associational rule

IF conclude(acute-meningitis no .7)

THEN conclude(viral-meningitis .5)

since acute is a sub-type of viral-meningitis.

Viral-meningitis is on the differential.

(i.e. prior information has activated

viral-meningitis as a potential hypothesis.)

The suggested modification completes this explanation.

The following rule automatically generated and linked into the Neomycin knowledge base, along with the corresponding set of clauses that includes the relation evidence.for(photophobia acute.meningitis rule1 .7).

**Object-Level Rule 1.**

IF:        patient.finding(photophobia yes)

THEN:     conclude(acute.meningitis (cf = .7))

## CHAPTER 6

### INHERENT LIMITS OF LEARNING

Ideally, in the limit, an apprenticeship learning technique should create a perfect or optimal performance element. The knowledge base should reach a state where it is the best possible knowledge base for its associated performance element. At the very least, improvements to the knowledge base should lead to better problem-solving performance. In no case should performance be worsened by additional knowledge learned in an apprenticeship setting. Unfortunately, changes made to a knowledge base by an apprentice learning program do not always lead to better performance. This problem may be characteristic of most existing knowledge bases. This chapter describes some inherent limitations to improving a knowledge base via apprenticeship learning, and a non-apprenticeship approach that circumvents these limitations.

This chapter shows that interactive debugging techniques, which subsume apprenticeship learning techniques, are inherently inadequate for improving a large class of knowledge bases. These are denoted as *sociopathic* knowledge bases and they share the property that 'better' knowledge does not necessarily lead to a 'better' knowledge base. That is, problem-solving performance can deteriorate when additional good knowledge is added to a knowledge base. This chapter explains the nature of the inherent limits to debugging

such knowledge bases.

In Section 6.1 the concept of a sociopathic knowledge base is defined and shown to encompass many existing knowledge bases. In Section 6.2, interactive debugging techniques are shown to be inadequate for debugging such a knowledge base. In Section 6.3, the problem of reducing sociopaths is formalized and an exact solution method is shown to be NP-Complete. A heuristic sociopathic reduction algorithm is then presented and experimental results of the algorithm are described.

## 6.1 Sociopathic Knowledge Bases

**Definition 6.1** Let  $Perf(PS)$  be a performance metric for evaluating the accuracy of problem-solving performance. An EDB is *sociopathic* iff

1. All tuples in the EDB are correct.
2.  $\exists KB' \subset KB \mid Perf(PS(KB')) \gg Perf(PS(KB))$ .

◇

The EDB excludes facts that can be derived using rules of inference from tuples encoded in the EDB. An assumption is made that it is possible to determine whether each tuple is correct either by asking an expert or using an underlying theory of the domain as described in Chapter 5.

The idea of non-monotonic reasoning is similar to but different from the sociopathicity property. In non-monotonic reasoning, the addition of new facts to the knowledge base may invalidate conclusions reached using the knowledge base. In a sociopathic knowledge base, it is problem-solving performance that is non-monotonic with the addition of new facts.

Consider knowledge bases that contain probabilistic rules such as MYCIN-type CFs. A knowledge base that contains these types of rules will be a sociopathic knowledge base. This is because heuristic inference rules with a measure of strength less than certainty have an unusual property: better individual rules do not necessarily lead to a better overall rule set. All less-than-certain rules contribute evidence towards erroneous conclusions for some problem instances, and the distribution of these erroneous conclusions over the instances is not necessarily related to individual rule quality. This has important consequences for automatic machine learning of rules, since rule selection is usually based on measures of quality of individual rules.

An obvious and intuitively reasonable solution to this problem, incremental modification and deletion of rules responsible for wrong conclusions *a la* Teiresias, is not always appropriate. In empirical ODYSSEUS experiments, it failed to converge to an optimal set of rules. Given a set of heuristic rules, the best rule set should be considered to be the element of the power set of rules that yields a global minimum error with respect to generating erroneous positive and negative conclusions. In this chapter, this selection process is modeled as a bipartite graph minimization problem and shown to be NP-complete. A solution method is described, the Sociopathic Reduction Algorithm, that performs a model-directed search of the rule space. On an example from medical diagnosis, the Sociopathic Reduction Algorithm significantly reduced the number of misdiagnoses when applied to a rule set generated from 104 training instances.

### 6.1.1 Reasoning Under Uncertainty

Reasoning under uncertainty has been widely investigated in artificial intelligence. Probabilistic approaches are of particular relevance to rule-based expert systems, where one is interested in modeling the heuristic and evidential reasoning of experts. Methods

developed to represent and draw inferences under uncertainty include the certainty factors used in MYCIN (Buchanan and Shortliffe, 1984), fuzzy set theory (Zadeh, 1979), and the belief functions of Dempster-Shafer theory (Shafer, 1976) (Gordon and Shortliffe, 1985). In many expert system frameworks, such as EMYCIN, EXPERT, MRS, S.1 and KEE, the rule structure permits a conclusion to be drawn with varying degrees of certainty or belief. This chapter addresses a concern common to all these methods and systems.

In refining and debugging a probabilistic rule set, there are three major causes of errors: *missing* rules, *wrong* rules, and *deleterious interactions* between good rules. The purpose of this chapter is to explicate a type of deleterious interaction and to show that it (a) is indigenous to rule sets for reasoning under uncertainty, (b) is of a fundamentally different nature from missing and wrong rules, (c) cannot be handled by traditional methods for correcting wrong and missing rules, and (d) can be handled by the method described in this chapter. The method provides a way of compensating for the inherent limitations of apprenticeship learning.

Section 6.2 describes the type of deleterious rule interactions that have been encountered in connection with automatic induction of rule sets using ODYSSEUS, and explain why the use of most *rule modification methods* fails to grasp the nature of the problem. Section 6.3 discusses approaches to debugging and refining rule sets and explain why traditional *rule set debugging methods* are inadequate for handling global interactions. Section 6.4 formulates the problem of reducing deleterious interactions as a bipartite graph minimization problem and show that it is NP-complete. Section 6.5 presents a heuristic solution method called the Sociopathic Reduction Algorithm. Finally, experiences in using the Sociopathic Reduction Algorithm are described.

A brief description of terminology will be helpful to the reader. Assume there exists a collection of *training instances*, each represented as a set of feature-value pairs

of *evidence* and a set of *hypotheses*. Rules have the form  $LHS \rightarrow RHS (CF)$ , where LHS is a conjunction of evidence, RHS is a hypothesis, and CF is a certainty factor or its equivalent. A rule that correctly confirms a hypothesis generates *true positive* evidence; one that correctly disconfirms a hypothesis generates *true negative* evidence. A rule that incorrectly confirms a hypothesis generates *false positive* evidence; one that incorrectly disconfirms a hypothesis generates *false negative* evidence. False positive and false negative evidence can lead to *misdiagnoses* of training instances.

### 6.1.2 Inexact Reasoning and Rule Interactions

When operating as an evidence-gathering system (Buchanan and Shortliffe, 1984), an expert system accumulates evidence for and against competing hypotheses. Each rule whose preconditions match the gathered data contributes either positively or negatively toward one or more hypotheses. Unavoidably, the preconditions of probabilistic rules succeed on instances where the rule will be contributing false positive or false negative evidence for conclusions. For example, consider the following rule:<sup>1</sup>

$$\begin{aligned} & \text{finding}(\text{surgery, yes}) \wedge & & \text{(R1)} \\ & \text{hypothesis}(\text{gram\_neg\_infection, yes}) \rightarrow \\ & \text{conclude}(\text{klebsiella, yes, 0.77}) \end{aligned}$$

The frequency with which R1 generates false positive evidence has a major influence on its CF of 0.77, where  $-1 \leq CF \leq 1$ . Indeed, given a set of training instances, such as a library of medical cases, the certainty factor of a rule can be given a probabilistic interpretation<sup>2</sup> as a function  $\Phi(x_1, x_2, x_3)$ , where  $x_1$  is the fraction of the positive instances

<sup>1</sup> This is a simplified form of ( $\$And$  (Same Cntxt Surgery))  $\rightarrow$  (Conclude Cntxt Gram\_Negative\_Infection Klebsiella Tally 770)).

<sup>2</sup> See Appendix D for a description of the function  $\Phi$ . This statistical interpretation of CFs deemphasizes incorporating orthogonal utility measures as discussed in (Buchanan and Shortliffe,

of a hypothesis where the rule premise succeeds, thus contributing true positive or false negative evidence;  $x_2$  is the fraction of the negative instances of a hypothesis where the rule premise succeeds, thus contributing false positive or true negative evidence; and  $x_3$  is the ratio of positive instances of a hypothesis to all instances in the training set. For R1 in the MYCIN domain,  $\Phi(.43, .10, .22) = 0.77$ , because statistics on 104 training instances<sup>3</sup> yield the following values:

$$x_1: \text{LHS true among positive instances} = \frac{10}{23}$$

$$x_2: \text{LHS true among negative instances} = \frac{8}{81}$$

$$x_3: \text{RHS true among all instances} = \frac{23}{104}$$

Hence, R1 generates false positive evidence on eight instances, some of which may lead to false negative diagnoses. But whether they do or not depends on the other rules in the system; hence the emphasis on taking a global perspective. The usual method of dealing with situations such as this is to make the rule fail less often by specializing its premise (Michalski, 1983). For example, surgery could be specialized to neurosurgery, and R1 could be replaced with:

```
finding(neurosurgery, yes) ^ (R2)
hypothesis(gram_neg_infection, yes) →
conclude(klebsiella, Yes, 0.92)
```

On the case library of training instances for the R2 rule,  $\Phi(.26, .02, .22) = 0.92$ , so R2 makes erroneous inferences in two instances instead of eight. Nevertheless, modifying

---

1984).

<sup>3</sup> This chapter uses the MYCIN case library of 104 cases, and was written before the NEOMYCIN case library of 114 cases that is described in other chapters was constructed. These vocabulary of data requests and diseases in these two libraries is different.

R1 to be R2 on the grounds that R1 contributes to a misdiagnosis is not always appropriate; there are three objections to this frequent practice. First, both rules are *inezact* rules that offer advice in the face of limited information, and their relative accuracy and correctness is explicitly represented by their respective CFs. They are expected to fail, hence failure should not necessarily lead to their modification. Second, all probabilistic rules reflect a trade-off between generality and specificity. An overly general rule provides too little discriminatory power, and an overly specific rule contributes too infrequently to problem solving. A policy on proper grain size is explicitly or implicitly built into rule induction programs; this policy should be followed as much as possible. Specialization produces a rule that usually violates such a policy. Third, if the underlying problem for an incorrect diagnosis is rule interactions, a more specialized rule, such as the specialization of R1 to R2, can be viewed as creating a potentially more dangerous rule. Although it only makes an incorrect inference in two instead of eight instances, these two instances will be now harder to counteract when they contribute to misdiagnoses because R2 is stronger. Note that a rule with a large CF is more likely to have its erroneous conclusions lead to misdiagnoses. This perspective motivates the prevention of misdiagnoses in ways other than the use of rule specialization or generalization.

Besides rule modification, another way of nullifying the incorrect inference of a rule in an evidence-gathering system is to introduce counteracting rules. In the example, this would be rules with a negative CF that concludes *Klebsiella* on the false positive training instances that lead to misdiagnoses. But since these new rules are probabilistic, they introduce false negatives on some other training instances, and these may lead to misdiagnoses. Still more counteracting rules could be added with with a positive CF to nullify any problems caused by the original counteracting rules, but these rules introduce false positives on yet other training instances, and these may lead to other misdiagnoses.

Clearly, adding counteracting rules may not be necessarily the best way of dealing with misdiagnoses made by probabilistic rules.

## 6.2 Debugging Sociopathic Knowledge Bases

Assume the existence of a set of probabilistic rules that were either automatically induced from a set of training cases or created manually by an expert and knowledge engineer. In refining and debugging this probabilistic rule set, there are three major causes of errors: missing rules, wrong rules, and unexpected interactions among good rules. The types of rule interactions are first described, and then it is shown how the traditional approach to debugging is inadequate.

### 6.2.1 Types of Rule Interactions

There are many types of rule interactions in a rule-based system. Rules interact by *chaining* together, by using *the same evidence* for different conclusions, and by drawing the *same conclusions* from different collections of evidence. Thus one of the lessons learned from research on MYCIN (Buchanan and Shortliffe, 1984) was that complete modularity of rules is not possible to achieve when rules are written manually. An expert uses other rules in a set of closely interacting rules in order to define a new rule, in particular to set a CF value relative to the CFs of interacting rules.

Automatic rule induction systems encounter the same problems. Moreover, automatic systems lack an understanding of the strong semantic relationships among concepts to allow judgments about the relative strengths of evidential support. Instead, induction systems use *biases* to guide the rule search (Utgoff, 1986; Michalski, 1983). Examples of some biases used by the induction subsystem of the ODYSSEUS apprenticeship learning program are rule generality, whereby a rule must cover a certain percentage of instances;

rule specificity, whereby a rule must be above a minimum discrimination threshold; rule colinearity, whereby rules must not be too similar in classification of the instances in the training set; and rule simplicity, whereby a maximum bound is placed on the number of conjunctions and disjunctions (Wilkins et al., 1986).

### 6.2.2 Traditional Methods of Debugging a Rule Set

The standard approach to debugging a rule set consists of iteratively performing the following steps:

- Step 1. Run the system on cases until a false diagnosis is made.
- Step 2. Track down the error and correct it, using one of five methods pioneered by TEIRESIAS (Davis, 1982) and used by knowledge engineers generally:
  - Method 1: Make the preconditions of the offending rules more specific or sometimes more general.<sup>4</sup>
  - Method 2: Make the conclusions of offending rules more general or sometimes more specific.
  - Method 3: Delete offending rules.
  - Method 4: Add new rules that counteract the effects of offending rules.
  - Method 5: Modify the strengths or CFs of offending rules.

This approach may be sufficient for correcting wrong and missing rules. However, it is flawed from a theoretical point of view, with respect to its sufficiency for correcting problems resulting from the global behavior of rules over a set of cases. It possesses two

---

<sup>4</sup> Ways of generalizing and specializing rules are nicely described in (Michalski, 1983). They include dropping conditions, changing constants to variables, generalizing by internal disjunction, tree climbing, interval closing, exception introduction, etc.

serious methodological problems. First, using all five of these methods is not necessarily appropriate for dealing with global deleterious interactions. Section 6.2 explained why in some situations modifying the offending rule or adding counteracting rules leads to problems, and misses the point of having probabilistic rules, and this eliminates methods 1, 2 and 4. If rules are being induced from a training set of cases, modifying the strength of the rule is illegal, since the strength of the rule has a probabilistic interpretation, being derived from frequency information derived from the training instances, and this eliminates method 5. Only method 3 is left to cope with deleterious interactions. The second methodological problem is that the traditional method picks an arbitrary case to run in its search for misdiagnoses. Such a procedure will often not converge to a good rule set, even if modifications are restricted to rule deletion. Example 1 in Section 6.3.3 illustrates this situation.

The perspective on this topic that is presented here evolved in the course of ODYSSEUS experiments in induction and refinement of knowledge bases. Using 'better' induction biases did not always produce rule sets with better performance, and this prompted investigating the possibility of global probabilistic interactions. The original ODYSSEUS approach to debugging was similar to the TEIRESIAS approach. But often, correcting a problem led to other cases being misdiagnosed, and in fact this type of *automated* incremental debugging seldom converged to an acceptable set of rules. It might have if the common practice of 'tweaking' the CF strengths of rules was used. However this was not permissible, since our CF values have a precise probabilistic interpretation.

### 6.3 Minimizing Sociopathic Interactions

Assume there exists a large set of training instances, and a rule set for solving these instances has been created manually or by induction that is fairly complete and

contains rules that are individually judged to be good. By good, is meant that they individually meet some predefined quality standards such as the biases described in Section 6.1. Further, assume that the rule set misdiagnoses some of the instances in the training set. Given such an initial rule set, the problem is to find a rule set that meets some optimality criteria, such as to minimize the number of misdiagnoses without violating the goodness constraints on individual rules.<sup>5</sup> Now modifications to rules, except for rule deletion, generally break the predefined goodness constraints. And adding other rules is not necessarily desirable, for if they satisfied the goodness constraints they would have been in the original rule set produced, especially if the rule set was produced by an induction program. Hence, if a solution is to be found that meets the described constraints, the solution must be a subset of the original rule set.<sup>6</sup>

The best rule set is the element of the power set of rules in the initial rule set that yields a global minimum weighted error. A straightforward approach is to examine and compare all subsets of the rule set. However, the power set is almost always too large to work with, especially when the initial set has deliberately been generously generated. The selection process can be modeled as a bipartite graph minimization problem as follows.

### 6.3.1 Bipartite Graph Minimization Formulation

For each hypothesis in the set of training instances, define a directed graph  $G(V, A)$ , with its vertices  $V$  partitioned into two sets  $I$  and  $R$ , as shown in Figure 6.3.1. Elements of  $R$  represent rules, and the evidential strength of  $R_j$  is denoted by  $\Phi_j$ . Each vertex in  $I$  represents a training instance; for positive instances  $\Psi_i$  is 1, and for negative instances

---

<sup>5</sup> In META-DENDRAL, a large initial rule set was created by the RULEGEN program, which produced plausible individual rules without regard to how the rules worked together. The RULEMOD program selected and refined a subset of the rules. See (Buchanan and Mitchell, 1978) for details.

<sup>6</sup> If it is discovered that this solution is inadequate, then introducing rules that violate the induction biases is justifiable.

$\Psi_i$  is  $-1$ . Arcs  $[R_j, I_i]$  connect a rule in  $R$  with the training instances in  $I$  for which its preconditions are satisfied; the weight of arc  $[R_j, I_i]$  is  $\Phi_j$ . The weighted arcs terminating in a vertex in  $I$  are combined using an evidence combination function  $\Phi'$ , which is defined by the user. The combined evidence classifies an instance as a positive instance if the combined evidence is above a user specified threshold  $CF_t$ . In the example in section 6.1,  $CF_t$  is 0, while for MYCIN,  $CF_t$  is 0.2.

More formally, assume that  $I_1, \dots, I_m =$  training set of instances, and  $R_1, \dots, R_n =$  rules of an initial rule set. Then the function to be minimized is:

$$z = \sum_{j=1}^n b_j r_j$$

subject to the constraints

$$\bigwedge_{i=1}^m \left( \Phi' (a_{i1}r_1, \dots, a_{in}r_n) \otimes_i CF_t \right)$$

$$\sum_{j=1}^n r_j \geq R_{min}$$

where

$r_j =$  if  $R_j$  is in solution rule set then 1 else 0;

$b_j =$  bias constant to preferentially favor rules;

$a_{ij} =$  if arc  $[R_j, I_i]$  exists then  $\Phi_j$  else 0;

$CF_t =$  the CF threshold for positive classification;

$\Phi' =$  n-ary function for combining CFs, where

the time to evaluate is polynomial in  $n$ ;

$R_{min} =$  minimum number of rules in solution set;

$\otimes_i =$  if  $\Psi_i$  is 1 then " $>$ " else " $\leq$ ".

The solution formulation solves for  $r_j$ ; if  $r_j = 1$  then rule  $R_j$  is in the final rule set. The main task of the user is setting up the  $a_{ij}$  matrix, which associates rules and instances and indicates the strength of the the associations. Note that the value of  $a_{ij}$  is zero if the preconditions of  $R_j$  are not satisfied in instance  $I_i$ . Preference can be given to particular rules via the bias  $b_j$  in the objective function  $z$ . For instance, the user may wish to favor the selection of strong rules. The  $R_{min}$  constraint forces the solution rule set to be above a minimum size. This prevents finding a solution that is too specialized for the training set, giving good accuracy on the training set but having a high variance on other sets, which would lead to poor performance.

**Theorem 1.** The bipartite graph minimization problem for heuristic rule set optimization is NP-complete.

**Proof.** To show that the bipartite graph minimization problem is NP-complete, reduction from Satisfiability is used. Satisfiability clauses are mapped into graph instance nodes and the atoms of the clauses are mapped into rule nodes. Arcs connect rule nodes to instance nodes when the respective literals appear in the respective clauses. The evidence combination function ensures that at least one arc goes into each clause node from a rule node representing a true literal. The evidence combination function also performs bookkeeping functions.  $\diamond$

### 6.3.2 Sociopathic Reduction Algorithm

In this section, a solution method called the Sociopathic Reduction Algorithm is described, and an example is provided based on the graph shown in Figure 6.3.3. An alternative solution method uses zero-one integer programming. It is more robust, but places a restriction on the evidence combination function, namely that the evidence be additively combined. It is not adequate when using the certainty factor model, but may

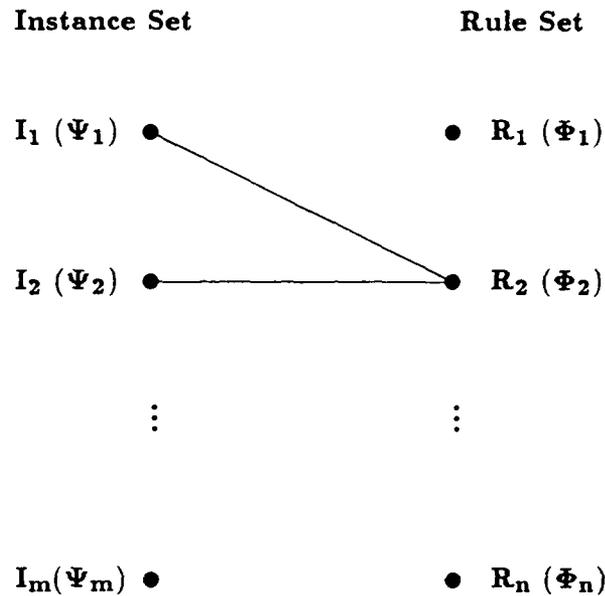


Figure 6.1: Bipartite graph formulation of sociopathic interactions. The instance set contains a set of problem descriptions. The rule set contains the rules in the knowledge base. The arcs show which rules apply to which cases. All arcs emanating from a rule node have the same weight, namely,  $\Phi$ .

be suitable for connectionist approaches.

The following model-directed search method, the Sociopathic Reduction Algorithm, is one that has been developed and used in ODYSSEUS experiments:

- Step 1. Assign values to penalty constants. Let  $p_1$  be the penalty assigned to a poison rule. A *poison rule* is a strong rule giving erroneous evidence for a case that cannot be counteracted by the combined weight of all the rules that give correct evidence. Let  $p_2$  be the penalty for contributing false positive evidence to a misdiagnosed case,  $p_3$  be the penalty for contributing false negative evidence to a misdiagnosed case,  $p_4$  be the penalty for contributing false positive evidence to a correctly diagnosed case,  $p_5$  be the penalty for contributing false negative evidence to a correctly diagnosed case, and  $p_6$  be the penalty for using weak rules. Let  $h$  be the maximum number of rules that are removed at each iteration. Let  $R_{min}$  be the minimum size of the

solution rule set.

- Step 2. Optional step for very large rule sets: given an initial rule set, create a new rule set containing the  $n$  strongest rules for each case.
- Step 3. Find all misdiagnosed cases for the rule set. Then collect and rank the rules that contribute evidence toward these erroneous diagnoses. The rank of rule  $R_j$  is

$\sum_{i=1}^6 p_i n_{ij}$ , where:

- $n_{1i} = 1$  if  $R_j$  is a poison rule or its deletion leads to the creation of another poison rule and 0 otherwise.
- $n_{2j} =$  the number of misdiagnoses for which  $R_j$  gives false positive evidence;
- $n_{3j} =$  the number of misdiagnoses for which  $R_j$  gives false negative evidence;
- $n_{4j} =$  the number of correct diagnoses for which  $R_j$  gives false positive evidence;
- $n_{5j} =$  the number of correct diagnoses for which  $R_j$  gives false negative evidence;
- $n_{6j} =$  the absolute value of the CF of  $R_j$ ;
- Step 4. Eliminate the  $h$  highest ranking rules.
- Step 5. If the number of misdiagnoses begins to increase and  $h \neq 1$ , then  $h \leftarrow h - 1$ .

Repeat steps 3-4 until either

- there are no misdiagnoses
- $R_{min}$  is reached
- $h = 1$  and the number of misdiagnoses begins to increase.  $\diamond$

Each iteration of the algorithm produces a new rule set, and each rule set must be rerun on all training instances to locate the new set of misdiagnosed instances. If this is particularly difficult to do, the  $h$  parameter in step 4 can be increased, but there is the

potential risk of converging to a suboptimal solution. For each misdiagnosed instance, the automated reasoning system that uses the rule set must be able to explain which rules contributed to a misdiagnosis. Hence, a system with good explanation capabilities is desirable.

The nature of an optimal rule set differs between domains. Penalty constants,  $p_i$ , are the means by which the user can define an optimal policy. For instance, via  $p_2$  and  $p_3$ , the user can favor false positive over false negative misdiagnoses, or visa versa. For medical expert systems, a false negative is often more damaging than a false positive, as false positives generated by a medical program can often be caught by a physician upon further testing. False negatives, however, may be sent home, never to be seen again.

In ODYSSEUS experiments, the value of the six penalty constants was  $p_i = 10^{6-i}$ . The  $h$  constant determines how many rules are removed on each iteration, with lower values, especially  $h \leq 3$ , giving better performance.  $R_{min}$  is the minimum size of the solution rule set; its usefulness was described in Section 6.1.

### 6.3.3 Example of Sociopathic Reduction

In this example, which is illustrated in Figure 6.3.3, there are six training instances, classified as positive or negative instances of the hypothesis. There are five rules shown with their CF strength. The arcs indicate the instances to which the rules apply. To simplify the example, define the combined evidence for an instance as the sum of the evidence contributed by all applicable rules, and let  $CF_i = 0$ . Rules with a CF of one sign that are connected to an instance of the other sign contribute erroneous evidence. Two cases in the example are misdiagnosed:  $I_4$  and  $I_5$ . The objective is to find a subset of the rule set that minimizes the number of misdiagnoses.

Assume that the final ruleset must have at least three rules, hence  $R_{min} = 3$ . Since

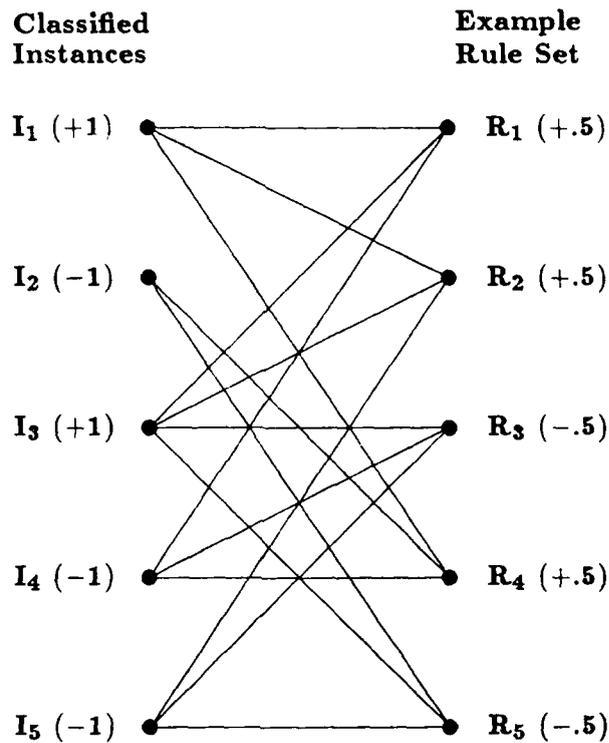


Figure 6.2: Example of bipartite graph formulation for one hypothesis. Each problem case instance is marked as a positive or negative example of a particular hypothesis. An arc from a rule node to an instance node means that the rule provides either positive or negative evidence for the hypothesis. The actual weight provided by the rule is shown next to the rule number.

all rules have identical magnitude and out degree, it is reasonable to set the bias to the same value for all  $n$  rules, hence  $b_j = 1$ , for  $1 \leq j \leq n$ . Let  $p_i = 10^{6-i}$ , for  $0 \leq i \leq 5$ , thus choosing rules in the highest category, and using lower categories to break ties.

On the first iteration, two misdiagnosed instances are found,  $I_4$  and  $I_5$ , and four rules contribute erroneous evidence toward these misdiagnoses,  $R_2$ ,  $R_3$ ,  $R_4$ , and  $R_5$ . Rules are ranked and  $R_4$  is chosen for deletion. On the second iteration, one misdiagnosis is found,  $I_4$ , and two erroneous rules contribute erroneous evidence,  $R_3$  and  $R_5$ . Rules are ranked and  $R_5$  is deleted. This reduces the number of misdiagnoses to zero and the algorithm successfully terminates.

The same example can be used to illustrate the problem of the traditional method of

rule set debugging, where the order in which cases are checked for misdiagnoses influences which rules are deleted. Consider a TEIRESIAS style program that looks at training instances and discovers  $I_4$  is misdiagnosed. There are two rules that contribute erroneous evidence to this misdiagnosis, rules  $R_3$  and  $R_5$ . It wisely notices that deleting  $R_5$  causes  $I_3$  to become misdiagnosed, hence increasing the number of misdiagnoses; so it chooses to delete  $R_3$ . However, no matter which rule it now deletes, there will always be at least one misdiagnosed case. To its credit, it reduced the number of misdiagnoses from two to one; however, it fails to converge to a rule set that minimizes the number of misdiagnoses.

#### 6.3.4 Experimental Results

Experiments with the Sociopathic Reduction Algorithm were performed using the MYCIN case library (Buchanan and Shortliffe, 1984). The ODYSSEUS experiments involved using 119 evidential findings, 26 intermediate hypotheses, and 21 final hypotheses. The training set had 104 training instances and each instance was classified as a member of four hypothesis classes on the average. The generated rules had one to three LHS conjuncts.

In ODYSSEUS experiments, approximately forty rule sets were generated containing between 200 and 20000 rules. Large rule sets were generated because of the interest in investigating the construction of knowledge bases that allow an expert system to automatically follow the line of reasoning of an expert; understanding a community of problem solvers requires more knowledge than that needed to just solve diagnosis problems. Typically, 85% of the training instances were diagnosed correctly, and seven out of ten cases used to evaluate the original MYCIN system were evaluated correctly. While ten cases is a small number for a validation set, it is a carefully constructed set and has been found adequate in accurately classifying human diagnosticians at all levels (Yu et al., 1979). Further, since there are an average of four hypotheses in the diagnosis per instance, the

training set can be viewed as having 416 instances and our validation set as having 40 instances. After the Sociopathic Reduction Algorithm was applied, 95% of the training instances was diagnosed correctly, and 80% of the validation set was diagnosed correctly.

Besides almost always converging to a solution in which all members of the training set are diagnosed correctly, the Sociopathic Reduction Algorithm is very efficient. Only eight to twelve iterations were required for rule sets created by ODYSSEUS that contained between 500 and 1450 rules. It was surprising to see how greatly performance is improved by deleting a small percentage of the rules in the rule set. As the results show, the improved performance on the training set carried over to the validation set.

## 6.4 Summary

Traditional methods of debugging a probabilistic rule set are suited to handling missing or wrong rules, but not to handling deleterious interactions between good rules. The underlying reason for this phenomenon have been described. The problem of minimizing deleterious rule interactions was formalized as a bipartite graph minimization problem and proved that it is NP-Complete. A heuristic method was described for solving the graph problem, called the Sociopathic Reduction Algorithm. In ODYSSEUS experiments, the Sociopathic Reduction Algorithm gave good results. It reduced the number of misdiagnoses on the training set from 15% to 5%, and the number of misdiagnoses on the validation set from 30% to 20%.

The rule set refinement method described in this chapter, or its equivalent, is an important component of any learning system for automatic creation of probabilistic rule sets for automated reasoning systems. All such learning systems will confront the problem of deleterious interactions among good rules, and the problem will require a global solution method, such as has been described here.

## CHAPTER 7

### UPPER LIMITS OF LEARNING

The ideal upper limit of learning in an apprentice setting is when all deficiencies in a knowledge base can be detected, and all detected deficiencies can be corrected. Since an apprenticeship is not a panacea for human experts, it is desirable to quantify its limits for an apprentice expert system.

A framework and procedure is presented in this chapter for calculating a *performance upper bound* for any apprenticeship learning technique that detects knowledge base deficiencies by modeling a human problem solver against a knowledge based system. This procedure, called the *synthetic agent method*, systematically explores the space of near-miss training instances because the learning apprentice observes a synthetic agent problem solver that uses a knowledge base that is minimally different from the knowledge base of the expert system. The synthetic agent method expresses the limits of debugging in terms of the knowledge representation and control language constructs of the expert system, namely the relation names of the EDB vocabulary.

The synthetic agent method involves the use of a *synthetic agent*: a copy of the expert system that initially has the exact same knowledge as the expert system. This chapter only presents an algorithm for evaluating an apprenticeship learning program

using the synthetic agent method; no experimental results are described.

It is worthwhile understanding the upper limit of apprenticeship learning to detect missing or wrong knowledge. It would be good to understand if there are certain types of errors that an apprenticeship is inherently unable to detect. This "limits of detection" contrasts with the subject of the last chapter, which was on "limits of correction"; repair techniques were shown to be appropriate for performance problems caused by missing or wrong knowledge, but inherently limited on problems caused by certain types of knowledge interactions.

We are especially interested in understanding the relative difficulty required in learning tuples from the approximately 120 different EDB relations in HERACLES. How successful can ODYSSEUS detect deficiencies caused by different relations? Is the absence of tuple from certain relations always noticeable? Are there particular types of knowledge whose absence is very hard to recognize? For example, HERACLES represents final diagnoses in a hierarchical tree structure; determining that a misdiagnosis is caused by a missing link in this structure, for example `parent(fungal-meningitis coccidioides)`, may be very difficult for the apprentice to discover. By contrast, it may be very easy to discover whether a trigger property of a rule is missing. A trigger property causes the conclusion of a rule to be treated as an active hypothesis if particular clauses of the rule premise are satisfied.

The performance upper limit calculated by the synthetic agent method identifies missing or erroneous knowledge in an intelligent agent that a particular differential modeling system is incapable of identifying. By contrast, most performance evaluation procedures aim to determine a performance lower bound; they experimentally demonstrate that a particular differential modeling system can successfully identify some missing or erroneous knowledge.

If ODYSSEUS fails to recognize the existence of a knowledge base deficiency this may be to to inherent limitation of debugging via differential modeling or simply a shortcoming of the ODYSSEUS component for detecting deficiencies. Note that deficiency detection is the responsibility of the apprentice in the machine learning scenario and the expert in the intelligent tutoring scenario.

This remainder of this chapter is organized as follows. Section 7.1 identifies important performance evaluation issues related to evaluation of a learning program. Section 7.2 presents and discusses the synthetic agent method. Finally, Section 7.3 describes an application of the synthetic agent method.

## 7.1 Evaluation and the Synthetic Agent Method

The synthetic agent method is considerably different from standard performance evaluation methods in two fundamental ways. First, since a single knowledge base should support multiple performance element's, evaluation criteria for a knowledge-based system should be quality of the individual knowledge tuples, not the quality of a single performance element. These metrics only partially overlap and certainly conflict in the short term. Second, the proposed synthetic agent method is to delineate a *performance upper bound*. A performance upper bound describes where and under what conditions a learning system for a problem solver must fail. By contrast, the standard evaluation approach aims at showing the extent to which a learning system can succeed. Further, instead of characterizing the limits of learning in terms of a percentage of problems that cannot be solved, the synthetic agent method characterizes the performance upper bound in terms of the knowledge representation language and the inference constructs used in the expert system, namely, the the EDB relations.

Another method of validating a learning program is to have the learning program

watch a student solve a training problem set. Let us assume that the student exhibits a representative set of the types of domain knowledge errors that could be made in the problem domain. A domain expert can manually identify the domain knowledge errors connected with each training problem. This manual analysis provides a performance upper bound with respect to this training set for the learning program, and the learning program is measured against this standard. When assuming that the student and the problem set involve all possible types of domain knowledge errors to be made, the goals of manual analysis and our proposed automated analysis using the synthetic agent method are identical.

We desire to know those types of differences in an expert system knowledge base that cannot be detected or corrected via differential modeling. In contrast to the capability-oriented approach, our validation approach aims at determining when the differential modeler must fail — we are *limitation-oriented*. For example, a limit of a program for inducing LISP functions from examples might be that the program can't induce cases that require certain types of loop constructs. In our work, we have focused on showing certain conditions that force the differential modeling approach to fail under the most favorable of conditions, the single fault assumption. The multiple fault assumption would allow determination of a broader performance upper bound.

## 7.2 Synthetic Agent Method of Evaluation

The apprenticeship learning scenario involves a human problem solver and an expert system. The *synthetic agent method* consists of replacing the person with a synthetic agent, which is another expert system, in order to experiment with and evaluate the apprenticeship learning system objectively. The knowledge in the synthetic agent expert system is modified to be slightly different from the knowledge in the original expert system.

This situation is illustrated in Figure 7.1.

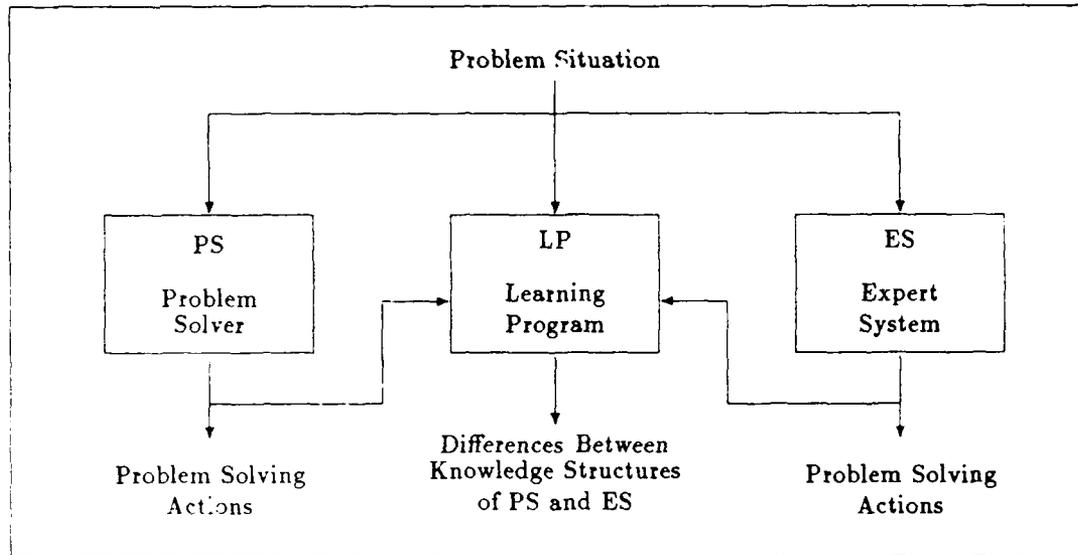


Figure 7.1: The synthetic agent for the learning by watching scenario. The human problem solver (PS) is replaced by a copy of the expert system (ES), and PS is observed by the learning program (LP). When a tuple is deleted from ES and LP observes PS, it sees a synthetic expert. When a tuple is deleted from PS and LP observes PS, it sees a synthetic student.

Since the synthetic agent method involves two almost identical expert systems, some terminological conventions will prove very helpful. The original HERACLES-based expert system is referred to as ES. The copy of this expert system that replaces the human problem solver is denoted as PS. Finally, the learning program being evaluated will be referred to as LP. It always is watching the PS expert system.

An advantage of the synthetic agent method is control over interpersonal variables involved in modeling a human problem solver against an expert system. An example of an interpersonal variable is the problem-solving style of a PS, as exemplified by the set of strategic diagnostic operators used by the PS. Diagnostic operators specify the permissible task procedures that can be applied to a problem as well as the allowable methods for achieving the task procedures. Examples of problem-solving operators in the domain

of diagnosis include: ask.general.questions, ask.clarifying.questions, refine.hypotheses, differentiate.between.hypotheses, and test.hypothesis. Another interpersonal variable is the quantity of domain-specific knowledge that the PS possesses.

While control of interpersonal variables would lead to an incorrect LP performance lower limit, conclusions reached concerning a performance upper limit are sound when interpersonal variables are controlled. If a system is inherently limited under the most optimal assumptions possible for modeling, it will be limited in the same way in all less optimal settings.

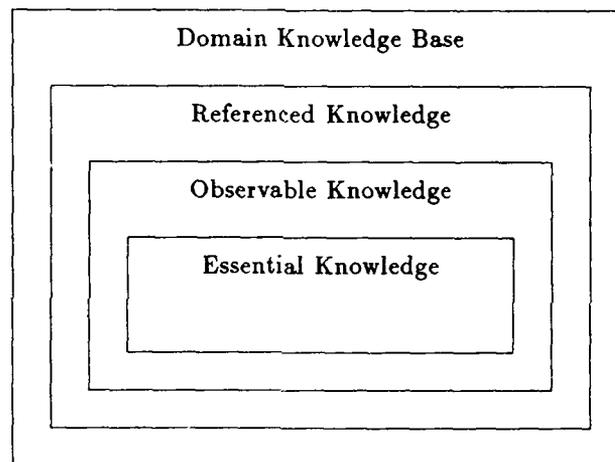


Figure 7.2: With respect to solving a particular problem, different part of the knowledge base are more important than others in solving the problem. This diagram shows the subset relationships between four important classes of knowledge.

In the learning and tutoring scenarios, the synthetic agent method treats the original expert system knowledge base of ES as a 'gold standard'. The apprentice ES and the student PS are engineered to have a deficiency with respect to this gold standard. In this chapter we restrict our analysis to the situation where the apprentice's knowledge differs from the gold standard by a single EDB tuple or all the tuples associated with a single rule; hence we have a single fault assumption. Two types of knowledge base deficiencies are

possible: missing knowledge and erroneous knowledge. The synthetic agent method procedure described in Section 7.2.1 shows how deletion of knowledge can represent the space of missing and erroneous knowledge. Other methods for creating erroneous knowledge are described in section 7.2.2.

For a given problem case, we distinguish between referenced, observable, and essential knowledge in the ES's knowledge base. The relation between these categories is illustrated in Figure 7.2. *Referenced knowledge* refers to tuples that are accessed during a problem solving case. *Observable knowledge* refers to tuples whose removal leads to different external observable behavior of a PS, either in the sequence of actions that the PS exhibits or the final answer. *Essential knowledge* refers to tuples whose removal leads to a significantly different final answer. Of most concern is the apprentice's ability to acquire the essential knowledge connected with a problem case, since this is the only knowledge that is critical to solving the case. For plausible reasoning systems, what comprises a significantly different answer needs to be specified. For instance, if there are multiple diagnoses, the significance of the order in which the hypotheses are ranked needs to be determined. Acquisition of knowledge that is observable but not essential is also of interest, since such knowledge can be essential for another problem case.

The procedure for calculating a performance upper limit for an apprenticeship system is now presented.

### 7.2.1 The Synthetic Agent Method

Step 1. Create synthetic agent. Replace PS with a synthetic agent: a copy of ES with initially the same object-level knowledge.<sup>1</sup>

---

<sup>1</sup> For expository purposes, this PS is called a *synthetic novice* in an intelligent tutoring scenario, a *synthetic expert* in a machine learning scenario, and a *synthetic agent* when any scenario may be meant. And to minimize confusion, the original ES is called the *original agent*.

Step 2. Solve problem case. Solve a problem using PS and save the *solution trace*, i.e., the observable actions of PS and the final answer.

Step 3. Identify observable knowledge. Collect all EDB tuples that were referenced by PS during the problem solving in Step 2. Identify the *observable knowledge*: the subset of the referenced tuples whose removal would lead to a different solution trace or a different final answer.

Step 4. For each observable tuple:

Step 4a. Remove the tuple from ES. In an knowledge acquisition scenario this creates an apprentice expert ES with *missing knowledge*. In an intelligent tutoring scenario this creates a student PS with *erroneous knowledge*. The tuple removed from the ES is declared to be erroneous<sup>2</sup>.

Step 4b. Detect and localize knowledge deficiency. Have the PS re-solve the case mentioned in Step 2. See if LP can detect the deficiency and suggest and evaluate the repair.

Step 5. For each observable tuple:

Step 5a. Remove the tuple from PS. In an intelligent tutoring scenario this creates a synthetic student PS with *missing knowledge*. In a knowledge acquisition scenario this creates an apprentice expert ES with *erroneous knowledge*. The tuple removed from the PS is declared to be erroneous<sup>2</sup>.

Step 5b. Detect and localize knowledge deficiency. Have the PS re-solve the case mentioned in Step 2. See if LP can detect the deficiency and suggest and evaluate a repair.

---

<sup>2</sup> This tuple of knowledge is treated as erroneous for purposes of validation. In reality, the tuple is true knowledge.

### 7.2.2 Discussion of Synthetic Agent Method

An expert system's explanation facility can be helpful in locating the observable knowledge, for a given problem case. One of the hallmarks of a good expert system is its ability to explain its own reasoning. So it is not too much to ask for those pieces of knowledge used on a problem case, and a good explanation system might even be able to identify the essential knowledge. At worst, given the pieces of knowledge that were used to solve a particular problem, the essential pieces of knowledge can be determined by experimentation. Usually, only a small amount of an expert system's domain knowledge is observable with respect to a given problem; and our experiences in the medical diagnosis domain have shown us that only a small amount of the observable knowledge is essential knowledge.

Some knowledge that is referenced by the expert system may not have observable consequences, even if it is used by the problem solver, since the a problem solver. For instance, in MYCIN and NEOMYCIN, terms that removal of knowledge does not always effect the external behavior of represent medical symptoms and measurements, such as patient weight, have an ASKFIRST property. The expert system uses the value of this property to decide whether the value of a variable is first determined by asking the user or first determined by derivation by some other method, such as from first principles. However, if the system does not possess techniques for deriving the information from other principles, then the external behavior of the system is the same regardless of the value of the ASKFIRST property.

To determine which relations are most easily learned, one would consider each EDB relation separately and determine by analysis whether the relation is always observable, only sometimes observable, or never observable. One would then find a problem case where tuples from that relation are essential knowledge, or demonstrate that no such problem

case case exist. Then apply the synthetic agent procedure given above.

When testing the detection of knowledge base deficiencies in steps 4b and 5b of the synthetic agent method, part of the assessment must relate to whether the apprentice detects knowledge base differences close to the point in the problem-solving session where the different knowledge was used. This temporal proximity is important, since the problem-solving context at this point in the problem-solving session strongly focuses the search for missing or erroneous knowledge.

### 7.2.3 Categories of Errors

The knowledge organization that we focus upon specifies all object-level knowledge in a declarative fashion. In such a knowledge base, there are two main categories of errors: missing and erroneous knowledge. Missing knowledge is absent from the knowledge base, and erroneous knowledge is factually incorrect knowledge that is present in the knowledge base.

The space of missing knowledge is easy to generate, especially with the single fault assumption. Recall that the original expert system serves as our gold standard and the domain knowledge in the expert system is declaratively represented. Hence, the number of single faults from missing knowledge is equal to the number of EDB tuples plus the number of rules.

The space of erroneous knowledge is much more difficult to describe. The synthetic agent method takes a novel approach to the problem in steps 5a and 6a. An erroneous tuple is created by declaring a correct tuple to be erroneous for purposes of validation. Erroneous knowledge can also be generated by substituting different but plausible terms for the correct tuple.

### 7.3 Summary

With the proliferation of expert systems, methods of intelligent tutoring and apprenticeship learning that are based on modeling the normal problem solving behavior of a student or expert against a knowledge-intensive expert system should become increasingly common. The synthetic agent method provides an objective means of assessing the limits of a particular apprenticeship program in the context of intelligent tutoring and apprenticeship learning. The power of an apprentice system is crucially dependent upon the expert system's method of knowledge representation and control. The synthetic agent method provides a means of expressing the limitations of a apprentice system in terms of the knowledge representation and control vocabulary.

What prior knowledge is required to learn in an apprenticeship setting? What types of knowledge are easy or difficult to learn? How well can ODYSSEUS follow the line of reasoning of human experts and human novices? How does the performance of ODYSSEUS compare to human apprenticeship learning performance? Clearly, an apprenticeship setting is not a panacea for human experts and we would like to quantify its limits for an apprentice learning program. The representation of knowledge in terms of EDB *relations* provides a framework for posing and answering such questions.

The synthetic agent method involves using an expert system as the synthetic problem solver to be observed. The synthetic problem solver's knowledge base has one more or one less piece of domain knowledge than the principal expert system, thereby creating a synthetic expert or a synthetic novice, respectively. The objective of future planned synthetic agent experiments is to determine whether the ODYSSEUS learning system can detect and localize the spectrum of potential knowledge differences in an apprenticeship learning setting. The synthetic agent method allows very controlled experiments to be performed, and thereby permits the establishment of a *performance upper bound*. That

is, what cannot be learned when using a synthetic agent is inherently unlearnable in an apprenticeship learning setting. We hope to express the inherent unlearnability results in terms of the EDB relations; it would be insightful to understand which ones are easy or difficult to learn. The synthetic agent method should place validation of an apprenticeship learning system on a more principled scientific footing and lead to a general methodology for evaluating apprentice learning systems in both a knowledge acquisition and student modeling context.

## CHAPTER 8

### LOWER LIMITS OF LEARNING

So far, we have focused on the interesting issue of what apprentice learning, in general, and our apprentice learning program, in particular, cannot accomplish, i.e., the inherent and upper limits of learning. However, it is important to determine not just what an apprenticeship cannot accomplish, but also what it can accomplish, and this is the subject of this chapter.

The *lower limit of learning* is the minimal degree of improvement in a performance element that is effected by a learning system. By its very nature, the lower limit is expressed with reference to a particular knowledge base and set of validation set of problems. A lower limit can be established by demonstrating that a certain level of performance improvement is achieved. It can be represented as the percentage of improvement in the performance element, on a scale from  $-1$  to  $+1$ . The ultimate lower limit is when a learning program modifies a perfect performance element so that it is incapable of solving any problem correctly, and this rates a  $-1$ . A lower limit of  $0$  is not dishonourable when the initial performance element works perfectly. Almost all prior validation experiments on learning programs for expert systems have concentrated on determination of a lower limit.

Performance evaluation of a learning program requires measuring the ability of a

performance element. The function of a learning program is to improve problem solving performance, and so this performance must be evaluated both before and after the apprenticeship learning session. Evaluation of the performance element of an expert system is a well-understood but difficult and time consuming task. Examples of performance evaluation studies based on a sound methodology are the evaluations of the MYCIN, INTERNIST and RL expert systems (Yu et al., 1979; Miller et al., 1984; Fu and Buchanan, 1985).

A typical way of measuring the degree to which a learning program improves a performance element is to use disjoint validation and training problem sets. First, one records the success rate of the problem solver at solving the *validation* problem set. Then the learning program modifies the performance element while watching a human expert solve a *training* problem set. Finally the performance element solves the validation problems again; the amount of improvement in performance on the validation problems provides a measure of the quality of the learning program.

This scenario establishes a lower bound on the quality of a learning program. By increasing the size of the training problem set, the learning program might improve the performance element even more. We refer to validation methods that establish a lower bound on the quality of a learning program as *capability-oriented*. For a given set of training and validation problems, capability-oriented validation shows that the learning program is responsible for a more capable performance element.

In apprenticeship learning this amounts to determining how often the learning program recognizes the situations in which the performance element's knowledge is deficient, and determining how well it corrects these deficiencies. In intelligent tutoring, validation of apprenticeship learning amounts to determining how often the learning program recognizes when the student's knowledge is deficient and identifies the deficiency. We will now present our experimental results in these two areas.

## 8.1 Knowledge Acquisition Results

Our knowledge acquisition experiments centered on improving the knowledge base of the NEOMYCIN expert system for diagnosing neurology problems. The NEOMYCIN vocabulary includes sixty diseases, but our physician John Sotos, determined that the existing data request vocabulary only allowed diagnosis of ten of these diseases. Another physician, Edward Herskovits, constructed a case library of 115 cases for these ten diseases from actual patient cases from the Stanford Medical Hospital, to be used as for testing ODYSSEUS. The validation set consisted of 112 of these cases. The most recent version of NEOMYCIN, version 2.3, initially diagnosed 31.25% of these cases correctly.

For use as a training set, problem solving protocols were collected of John Sotos solving six cases. The two longest protocol sessions were input to ODYSSEUS, to test it's ability in a learning by watching apprenticeship setting. The input to ODYSSEUS for each data request included the physician's focus for each data request, which provides information on the physician's high-level goal.

ODYSSEUS learned 16 rules from watching these two cases. These rules were found because they allowed metarule premises that contained *evidence for tuples to succeed*, thus completing a partial explanation of the expert's reasoning. Eight of these rules, those with a positive certainty factor, were added to the NEOMYCIN knowledge base of 152 rules, along with two data requests that were judged as 'general questions'; these are questions that should be asked of every patient. The negative rules were not added because NEOMYCIN does not work properly if rules with a negative CF are used.

The set of 112 cases was rerun, and NEOMYCIN solved 43.75% of the cases correctly. This represents over a 40% improvement in performance. John Sotos found all of the rules to be plausible medical knowledge, except for one, that linked aphasia to brain abscess. The performance of NEOMYCIN before and after learning is shown in Tables 8.1 and 8.2.

<i>Disease</i>	<i>Number Cases</i>	<i>True Positives</i>	<i>False Positives</i>	<i>False Negatives</i>
Brain Abscess	7	0	0	7
Bacterial Meningitis	16	16	47	0
Viral Meningitis	11	4	5	7
Fungal Meningitis	8	0	0	8
TB Meningitis	4	1	0	3
Cluster Headache	10	0	0	10
Tension Headache	9	9	20	0
Migraine Headache	10	1	1	9
Brain Tumor	16	0	0	16
Subarachnoid Hemorrhage	21	4	0	17
None	0	0	4	0
Totals	112	35	77	77

Table 8.1: Performance of NEOMYCIN before learning. There were 112 cases used in the validation set to test NEOMYCIN's performance. A misdiagnosis produces a false positive and a false negative.

The final results are shown in Table 8.3.

Compared to guessing by always selecting the disease that is *a priori* the most likely, the performance of the NEOMYCIN expert system is 3.44 standard deviations better. On

<i>Disease</i>	<i>Number Cases</i>	<i>True Positives</i>	<i>False Positives</i>	<i>False Negatives</i>
Brain Abscess	7	1	2	6
Bacterial Meningitis	16	12	31	4
Viral Meningitis	11	4	5	7
Fungal Meningitis	8	0	1	8
TB Meningitis	4	1	0	3
Cluster Headache	10	6	0	4
Tension Headache	9	9	11	0
Migraine Headache	10	2	0	8
Brain Tumor	16	5	3	11
Subarachnoid Hemorrhage	21	9	1	12
None	0	0	9	0
Totals	112	49	63	63

Table 8.2: Performance of NEOMYCIN after apprenticeship learning. This shows the results of NEOMYCIN after a learning by watching session using ODYSSEUS that involved watching a physician solve two medical cases.

a student-t test, this is significant at a  $t=.001$  level of significance. Thus we can conclude that NEOMYCIN's diagnostic performance is significantly better than guessing.

Compared to NEOMYCIN's baseline performance, the performance of NEOMYCIN

<i>Problem Solving Method</i>	<i>Correct Diagnoses</i>
Guessing randomly	10%
Guessing by always choosing most common disease	18%
NEOMYCIN before learning	31%
NEOMYCIN after learning: two cases watched	44%

Table 8.3: Performance improvement resulting from learning by watching. The first two entries show expected performance by guessing, with and without knowing the distribution of diseases. The last two entries show the performance of NEOMYCIN before and after the ODYSSEUS learning program modifies its knowledge base as the result of watching a physician diagnose two patients.

after improvement by ODYSSEUS is 2.86 standard deviations better. On a student-t test, this is significant for  $t = .006$ . The improved NEOMYCIN will perform better than the baseline NEOMYCIN in better than 99 out of 100 trials.

## 8.2 Student Modeling Results

Following the strategic reasoning of a student is a very difficult task. The student's LORE is often not in the ODYSSEUS LORE set because the NEOMYCIN knowledge base does is missing the domain knowledge necessary to make sense of the student's action. When the specialist being observed is NEOMYCIN, its actual LORE is always in the LORE set generated by ODYSSEUS for each data request. However, when observing second-year medical students, their LORE is in the LORE set of ODYSSEUS only 25% of the time. This

incompleteness is mostly due to sparse domain knowledge. Indeed, experiments with the MYCIN case library suggest that a four-fold expansion of the heuristic associational rules in NEOMYCIN would allow the student's LORE to be in the LORE-set 75% of the time. This was determined by analyzing data requests of students that are in the vocabulary of MYCIN and NEOMYCIN, where the hypotheses being pursued were in the vocabulary of MYCIN and NEOMYCIN. The ODYSSEUS induction system expanded the original set of MYCIN rules using induction over a library of MYCIN cases.

If determining a student's domain errors proves too difficult in a standard automatic programming situation, we suggest a novel approach to finding a student's misconceptions. The student will be requested to solve all 112 cases in the NEOMYCIN case library. Based on these problem solving sessions, the ODYSSEUS induction system will synthesize heuristic rules that correspond to the novice's beliefs. The synthesized rules can be compared against those generated by having an expert solve all of the cases. Note that not only does this allow finding a student's domain errors, but it provides a foundation for tracking the novice's strategic reasoning, and thus provides a potential means of identifying strategy errors. Because of the differences between the knowledge of a student and the knowledge of an expert system, it may be impossible in principle for a NEOMYCIN-based student modeling program to follow a student's strategic reasoning in the student protocols we collected when given just the data-requests. However, synthesis of the student's domain knowledge via the methods we have described reduce the number of domain differences between the knowledge of the student and the expert system, and thereby allow more focus on the discovery of strategy differences. Large differences in the domain knowledge causes a *cognitive tear* in the model (Burton and Brown, 1982). A cognitive tear occurs when a small amount of noise, introduced by the student's different knowledge, prevents a modeler from finding a coherent explanation of observed actions.

### 8.3 Comparing Apprentice Scenarios

There is a contrast between the two different ODYSSEUS apprenticeship scenarios of learning by watching and learning from experience. One way to compare these is to see how the two scenarios accomplish the three major learning tasks faced by an apprenticeship learning system: the realization that knowledge is missing, the generation of candidate repairs, and the testing of those repairs. Note that the last of these three is identical in both scenarios. On the other hand, detecting deficiencies is easier when watching oneself, because there is none of the uncertainty connected with inferring another agent's line of reasoning. Generating repairs is also easier when watching oneself, as there is no uncertainty as to exactly which metarule clause and hence which relation is responsible for the failure.

Compared to watching another problem solver, one can learn from watching one's own problem solving earlier in the knowledge acquisition 'end-game'. When watching another problem solver, a relatively large knowledge base is required; otherwise it is impossible to follow the line of reasoning of an expert most of the time, which is a requirement of this scenario.

A disadvantage of watching oneself is a large number of false alarms. Metarules fail most of the time, and it is not clear what the failure rate would be for a really good knowledge base. Perhaps it would only be a little lower than with a fairly incomplete knowledge base. More experimentation is required to answer these questions.

## CHAPTER 9

### CONCLUSIONS

The construction of expert system shells for generic tasks has become a common practice. There is a growing awareness that the power of a knowledge acquisition system for an expert system shell is bounded by the complexity and explicitness of the inference procedure or strategy knowledge used by the shell (Eshelman and McDermott, 1986; Kahn et al., 1985). There is also a growing awareness that *automated* knowledge acquisition must be grounded in an underlying domain theory (Mitchell et al., 1985; Smith et al., 1985). Using the HERACLES expert system shell and the ODYSSEUS apprenticeship learning program, we have demonstrated how underlying theories of a problem solving domain can be effectively used by a learning method centered around an explicit representation (e.g., tasks and metarules) of the problem solving method.

#### 9.1 Contributions

##### 9.1.1 Techniques for Apprenticeship Learning

The principal focus of this thesis has been *apprenticeship learning techniques* for expert systems. For human problem solvers, an apprenticeship is the most effective

method known for refining and debugging expertise in knowledge-intensive domains such as medicine and engineering; this motivates our research. An apprenticeship learning program, called ODYSSEUS, has been implemented; it improves the knowledge base for an expert system implemented with the HERACLES expert system shell.

Accomplishments include the development of a technique for learning by completing explanations that allows object-level knowledge to be learned by relaxing the constraints on the construction of an explanation. It has been demonstrated that the same technique of learning by completing explanations can transfer expertise in both directions: to and from a knowledge base.

This thesis demonstrated three scenarios for apprenticeship learning. The results show that an apprenticeship setting strongly focuses the learning; it strongly biases the learning process. In one experiment, a total of nine new rules and thirty new facts were learned by watching each of the forty steps of problem solving contained in two examples. These additions to the knowledge base improved the problem solving performance of an expert system by 40% on a suite of 112 cases. Our physician rated over 80% of the new knowledge as important and correct problem solving knowledge.

Two learning by watching scenarios were explored. The first involved having the expert provide an explanation of the motivation behind each problem solving action. This explanation was limited to the 'focus' each problem solving action. This goal information made tractable the problem of identifying knowledge base gaps. The second technique involved having the program determine the goal information. This latter method was problematic, as there are often multiple reasonable goals, thus masking learning opportunities.

We have developed a three stage model of end-game knowledge acquisition and shown how existing research contributes to each of the three stages. These three stages

are the detecting deficiencies, suggesting repairs, and evaluating repairs.

Improvements to the representation of strategy control knowledge in NEOMYCIN have been identified that are important for learning. These improvements relate to reversibility of the metarules, which is a key requirement of the ODYSSEUS approach. The problem solving state is recorded on property lists of LISP atoms; not explicitly recording all the variables that a metarule might change complicates reversibility. Finally, the use of a recursive control stack hinders backing up a computation to an arbitrary point. NEOMYCIN uses logic programming methods for implementation of knowledge representation but not for inference. These experiences suggest that logic programming would also aid inference.

### 9.1.2 Fundamental Limits of Learning

Another major focus of this dissertation was on fundamental and inherent limitations of learning techniques for knowledge-based expert systems. This section summarizes our results in this area.

Extant techniques of reasoning under uncertainty for expert systems are shown to lead to a *sociopathic knowledge base*. A knowledge base is sociopathic if there exists a subset of the knowledge base that gives better performance than the original knowledge base. Incremental learning techniques are shown to be *inherently limited* to improve an expert system with a sociopathic knowledge base.

The problem of minimizing sociopaticity was formalized as a bipartite graph minimization algorithm. This exact solution algorithm was proved to be NP-Complete. A heuristic method for removing sociopaticity was developed called the *Sociopathic Reduction Algorithm*; experiments show excellent convergence properties.

The *Synthetic Agent Method* was developed to determine the *upper performance*

*limits* of learning by observation for a knowledge-based expert system. The method systematically explores the space of near-miss training instances and expresses the results in terms of the learnability of the different types of knowledge in the knowledge base.

### 9.1.3 Symmetry of Learning and Teaching

Using the same mechanisms, ODYSSEUS semi-automates the transfer of knowledge *into* a knowledge base (learning) and *out of* a knowledge base (teaching). In the learning scenario, ODYSSEUS observes an expert, and functions as a knowledge acquisition program for the HERACLES expert system shell. In the teaching scenario, ODYSSEUS observes a student, and functions as a student modeling program for the GUIDON2 intelligent tutoring system, which is built over HERACLES. ODYSSEUS has been demonstrated in the domain of medical diagnosis, a task domain where apprenticeship learning plays a crucial role in the development of human experts.

The techniques developed for knowledge acquisition are not as effective for intelligent tutoring. The knowledge structures of an expert system more closely match those of an expert than a student. Following the problem solving steps of a student in terms of the expert system knowledge structures is more difficult.

## 9.2 Main Limitations of Approach

ODYSSEUS was validated by collecting problem solving protocols of over a dozen physicians and medical students, and feeding them to ODYSSEUS. The protocols were annotated by the human problem solvers to allow us to determine how well ODYSSEUS is able to follow the line of reasoning of human problem solvers, i.e., how well ODYSSEUS functions as an automated protocol analysis tool.

First, we discovered that the processes of detecting discrepancies and suggesting

repairs are underconstrained. That is, ODYSSEUS finds multiple good explanations for most actions of the expert and this is a problem since learning opportunities occur only when no explanations are found. We reduced these multiple interpretations by using information supplied by the human problem solver regarding the 'focus' for each data request. However, ideally, an apprentice system should be able to learn without using such information.

Second, for experts we have observed it appears that the NEOMYCIN knowledge base is too impoverished to follow the line of reasoning of experts when given just their data requests, and not focus information. That is, while NEOMYCIN is somewhat minimally competent to solve problems, this domain knowledge may not allow a modeling program to follow the line of reasoning of an expert. For each action of the expert, ODYSSEUS generates a set of explanations. The expert's correct explanation is not in this set 75% of the time. In most of these cases, analysis revealed that the reason is factual domain knowledge that is missing from NEOMYCIN. It is unclear that any reasoning based on 'patterns of interpretation' can be of help when the knowledge is so different. However, since the problem is missing domain knowledge, the solution to this problem might just be to add more knowledge so that the learning system can bootstrap itself. As was described in Section 8.2, some experiments with the MYCIN case library were performed that revealed that if the knowledge base was increased by a factor of four, this placed us at a bootstrapping threshold.

### 9.2.1 Underconstrained Interpretations of Actions

In this section, we describe the principal limitation of the ODYSSEUS approach to learning by watching, as well as a solution approach that might overcome this limitation.

The construction and testing of the ODYSSEUS program has revealed that the cen-

tral issue in detecting deficiencies in the knowledge base via apprenticeship learning is constraining the multiple interpretations of the actions of human problem solvers. Recall that apprenticeship learning is a form of failure-driven learning. A knowledge base discrepancy is suspected when no explanations can be construed for an observable action of the human problem solver. The generality of the ODYSSEUS approach causes the set of interpretations of problem solving actions and the set of conjectures of the underlying knowledge base discrepancy to be underconstrained.

The purpose of the ODYSSEUS subsystem that detects deficiencies is to select one or more LOREs as the expert's LORE or to decide that none of the LOREs provide an adequate explanation of the expert's action. Deciding there is no adequate LORE is very difficult, since there are often weakly plausible explanations for any action of the specialist; yet to learn, the program must recognize when none of its LOREs are sufficiently plausible.

An approach for future research that we believe might successfully constrain these multiple LOREs is as follows. The multiple LOREs should be shown to expert problem solvers in order to extract the heuristics they use to discard unlikely interpretations. Then an expert system should be created that uses these heuristics to prune the set of LOREs. Note that this formalizes the task of selecting among multiple interpretations as a problem that is solvable using the heuristic classification method; hence a HERACLES-based expert system should be appropriate to solve the multiple interpretations problem. In formalizing the detection of discrepancies as a classification problem, the pre-enumerated solution set is the set of LOREs.

The tasks of detecting discrepancies and suggesting repairs for apprenticeship learning are knowledge intensive tasks. There are three reasons why these functions should be formulated and implemented as HERACLES-based expert systems in their own right. First, these tasks are knowledge-intensive (Dietterich and Buchanan, 1981), and expert system

techniques are useful for representing large amounts of knowledge. Second, within an expert system architecture, the reasoning method used by the learning program can be made explicit and easily evaluated, since the domain knowledge is declaratively encoded using HERACLES' EDB relations for encoding domain knowledge. Third, ODYSSEUS can reason about any HERACLES-based system, then ODYSSEUS should be able to introspect and reason about its own knowledge and knowledge structures, and could theoretically improve itself in an apprenticeship learning setting. That is, it could debug and refine the knowledge base of the expert system for pruning multiple LOREs in an apprenticeship learning setting.

A variety of knowledge sources (KSs) would provide information required by the rules of the expert system for detecting deficiencies, thereby allowing it to rank LOREs. The more important knowledge sources are a *Heracles simulator KS*, *multiple interpretations KS*, *user model KS*, *strategic distance KS*, and a *patterns of interpretation KS*. A description of each of these KSs follows.

The *Heracles simulator KS* would process the information obtained during the problem solving session and this provides information on the current status of findings, hypotheses, and rules. Examples of current status information is whether the parameters and rules are known or unknown, true or false, and confirmed or disconfirmed. Rules for aiding the detecting of deficiencies relate parameters and rules to individual LOREs. For example, if the *Heracles simulator KS* believes that particular hypotheses have already been confirmed or disconfirmed, then there is a rule that attaches negative evidence to all LOREs that have as a goal or subgoal the confirmation one of these hypotheses.

The *multiple interpretations KS* would provide information that is used by rules for ranking and rating LOREs. The rules encode heuristics used by medical domain experts to arbitrate between multiple interpretations of a specialist's action. For instance, early

in the consultation session LOREs relating to more general hypotheses are preferred to LOREs with more specific hypotheses.

The *user model KS* would record user characteristics such as individual diagnostic style preferences, and these are used by rules for ranking LOREs. The rules arbitrate between competing action justifications. For example, some problem solvers have a depth-first problem-solving style, meaning that they pursue a particular hypothesis as soon as there is weak evidence confirming it. A rule would add support to those LOREs consistent with this style.

The *strategic distance KS* would provide information that aids in determining the similarity between HERACLES' preferred strategic action and the strategic action associated with each LORE. We assume that HERACLES' strategic knowledge exemplifies good strategic reasoning. Therefore, if the strategic action associated with a LORE is close to HERACLES' preferred strategic action, it is favored.

Finally, the *patterns of interpretation KS* would provide information that is used by rules that rank LOREs. The rules rate competing justifications according to the overall coherence they lend to the specialist's strategic plan.

If the expert system for detecting deficiencies is uncertain about an important fact, such as the expert's focus on a previous question in the consultation, the expert system can query the user, but this violates our goal of trying to automate the knowledge acquisition process as much as possible.

### 9.3 Further Work

Future research will continue in the area of apprenticeship learning. The method of detecting discrepancies must be improved: presently interpretations of expert behavior are underconstrained, and this slows down the *rate* at which learning proceeds.

There are many validation experiments yet to be performed. We would like to know the relative efficacy of the three apprenticeship learning scenarios.

Also, ODYSSEUS' techniques hold promise to tackle successfully important unsolved problems in the area of automatic programming from examples.

**APPENDICES**

## APPENDIX A

### ODYSSEUS PROGRAM

The ODYSSEUS system is implemented in Interlisp-D on a Xerox Dorado in the Knowledge Systems Lab of the Computer Science Department at Stanford University, and the Intelligent Systems Lab at Xerox Parc. This section describes the subsystems of ODYSSEUS that are implemented.

#### **Induction System (49 functions)**

Given a set of solved cases and a list of diagnostic hypotheses, Odys-kb-induce (ODYSSEUS knowledge base induce) creates a rule base for the HERACLES expert system shell. The program contains default induction biases that are easily modifiable. The program is being extended to provide the following capability: given a set of solved cases and a potential rule, determine the goodness of this rule in terms of induction biases and predictive accuracy.

#### **Explanation Generator and Recognizer (104 functions)**

Given a specialist solving a problem, Odys-diff-model (ODYSSEUS differential modeler) generates and ranks action justifications for each observed problem solving action of the specialist.

**Display System (59 functions)**

The display subsystem provides a graphics interface that allows a specialist to solve problems and justify actions using menus and a D-machine mouse.

**Knowledge Base Refinement System (15 functions)**

Given a rule set generated by Odys-kb-induce, this program finds a subset of the rules that has a global minimum weighted error. This program produces a solution in 12-25 iterations of the rule base.

## APPENDIX B

### HERACLES PROGRAM

The major domain knowledge base for HERACLES at this time is the NEOMYCIN knowledge base for diagnosing meningitis and neurological problems (Clancey, 1984). A second effort in the sand casting domain is called CASTER (Thompson and Clancey, 1986).

EDB tuples fall into five major classes, and the tuples for each class are listed in (Clancey and Bock, 1986). These classes are as follows.

- Static relations pertaining to constants and variables
- Static relations pertaining to rules
- Dynamic relations pertaining to beliefs
- Dynamic relations pertain to search and focus relations
- Dynamic relations with changing values

The first two classes encompass domain-specific knowledge. To learn a tuple in this class requires having a confirmation theory for this class. The relations that require a confirmation theory are as follows.

## Domain Relations Pertaining to Findings and Hypotheses

(SOFT-DATA \$FINDING)  
 (HARD-DATA \$FINDING)  
 (NONSPECIFIC \$FINDING)  
 (REDFLAG \$FINDING)  
 (STATE-CATEGORY \$HYP)  
 (TAXONOMIC \$HYP)  
 (PARENTOF \$TAXPARM \$PARENT)  
 (COMPLEX \$TAXPARM)  
  
 (CAUSES \$HYP1 \$HYP2)  
 (SUBSUMES \$FINDING1 \$FINDING2)  
 (PROCESSQ \$FINDING1 \$FINDING2)  
 (CLARIFYQ \$FINDING1 \$FINDING2)  
 (SOURCE \$FINDING1 \$FINDING2)  
 (SCREENS \$FINDING1 \$FINDING2)  
 (PROCESS-FEATURES \$HYP \$SLOT \$VAL \$FINDING)  
  
 (ALWAYS-SPECIFY \$FINDING)  
 (ASKFIRST \$FINDING)  
 (PROMPT \$FINDING \$VAL)  
  
 (BOOLEAN \$PARM)  
 (MULTIVALUED \$PARM)  
 (TABLE \$BLOCKPARM \$FINDING)  
  
 (ENABLINGQ \$HYP \$FINDING)  
 (SUGGESTS \$PARM \$HYP)  
 (TRIGGERS \$PARM \$HYP)

## Domain Relations Pertaining to Rules

(ANTECEDENT-IN \$FINDING \$RULE)  
 (APPLICABLE? \$RULE \$CNTXT \$FLG)  
 (EVIDENCEFOR? \$PARM \$HYP \$RULE \$CF)  
 (COMMONCASERULES \$HYP \$RULE)  
 (UNUSUALCASERULES \$HYP \$RULE)  
  
 (PREMISE \$RULE \$VAL)  
 (ACTION \$RULE \$VAL)  
 (ANTECEDENT \$RULE)  
 (TRIGGER \$RULE)  
 (SCREEN \$RULE)

## APPENDIX C

CALCULATING  $\Phi$ 

Consider rules of the form  $E \xrightarrow{CF} H$ . Then  $CF = \Phi = \Phi(x_1, x_2, x_3)$  = empirical predictive power of rule R, where:

- $x_1 = P(E^+|H^+) =$  fraction of the positive instances in which R correctly succeeds (true positives or true negatives)
- $x_2 = P(E^+|H^-) =$  fraction of the negative instances in which R incorrectly succeeds false positives or negatives
- $x_3 = P(H^+) =$  fraction of all instances that are positive instances

Given  $x_1, x_2, x_3$ , let

$$\bullet \quad x_4 = P(H^+|E^+) = \frac{x_1 x_3}{x_1 x_3 + x_2(1-x_3)}.$$

If  $x_4 > x_3$  then  $\Phi = \frac{x_4 - x_3}{x_4(1-x_3)}$  else  $\Phi = \frac{x_4 - x_3}{x_3(1-x_4)}$ .

This probabilistic interpretation reflects the modifications to the certainly factor model proposed by (Heckerman, 1986).

## APPENDIX D

## GLOSSARY

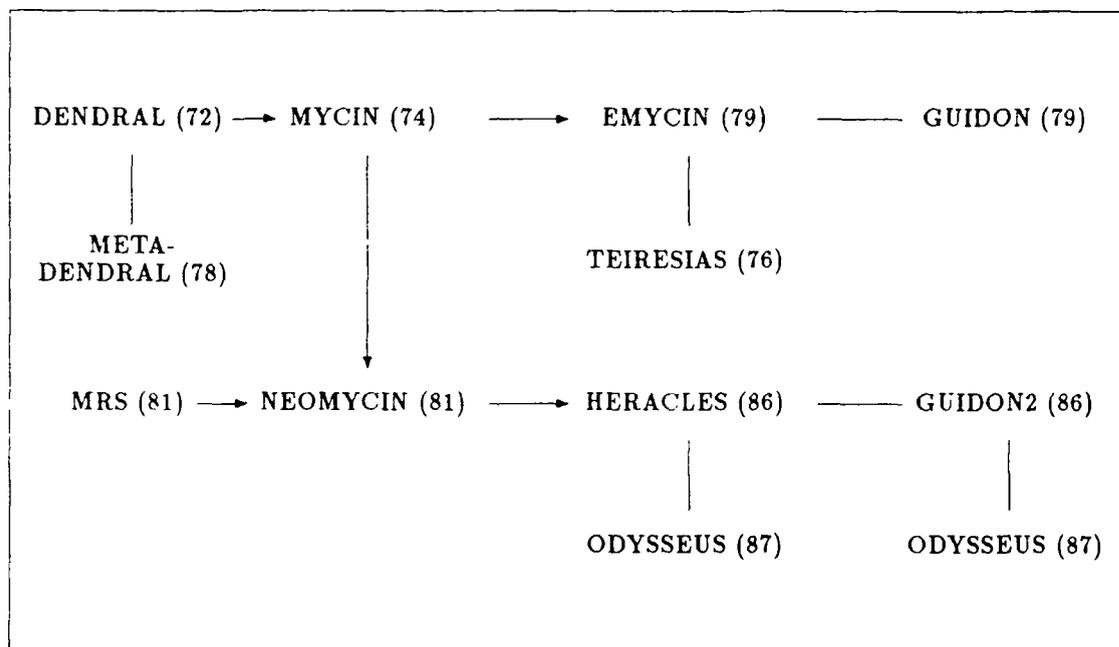


Figure D.1: Genealogy of the ODYSSEUS apprenticeship learning program. Only partial dependency links are shown. All chronologically earlier programs heavily influenced the design of all latter programs.

**DENDRAL.** An expert system for chemical structure determination (Buchanan and Feigenbaum, 1978; Lindsay et al., 1980).

**EMYCIN.** An expert system shell for classification-type problems consisting of the domain-independent part of MYCIN (VanMelle, 1980)

**GUIDON.** A case method tutor for transferring the expertise of a an EMYCIN knowledge base to a student (Clancey, 1979).

**GUIDON2.** An intelligent tutoring system for transferring the expertise of a HERACLES knowledge base to a student (Clancey, 1986).

**HERACLES.** An expert system shell for solving problems using the heuristic classification method, consisting of the domain-independent part of NEOMYCIN (Clancey, 1986).

**META-DENDRAL.** A knowledge acquisition program for the DENDRAL expert system (Buchanan and Feigenbaum, 1978; Lindsay et al., 1980).

**MYCIN.** An expert system for diagnosing meningitis and bacteremia infections (Shortliffe, 1976; Buchanan and Shortliffe, 1984).

**MRS.** A logic programming language, similar to PROLOG, augmented with meta-level control (Russell, 1985).

**NEOMYCIN.** A reworking of the MYCIN expert system. Strategy knowledge is explicitly represented as tasks and metarules. NEOMYCIN's domain is meningitis and diseases easily confused with meningitis (Clancey, 1984).

**ODYSSEUS.** A knowledge acquisition program for a HERACLES-based expert system, and a student modeling program for the GUIDON2 intelligent tutoring system (Wilkins et al., 1986; Wilkins, 1986).

**TEIRESIAS.** A knowledge acquisition program for an EMYCIN-based expert system  
(Davis, 1982).

## BIBLIOGRAPHY

## BIBLIOGRAPHY

- Amarel, S. (1986). Program synthesis as a theory formation task: problem representations and solution methods. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning, Volume II*, chapter 18, pages 499-570, Los Altos: Morgan Kaufmann.
- Anderson, J. (1983). Acquisition of proof skills in geometry. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An artificial intelligence approach*, chapter 7, pages 191-220, Palo Alto: Tioga Press.
- Barstow, D. R. (1979). *Knowledge-Based Program Construction*. New York: Elsevier-North Holland.
- Biermann, A. W. (1978). The inference of regular LISP programs from examples. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-8(8):585-600.
- Brown, J. S., Burton, R., and DeKleer, J. (1982). Pedagogical and knowledge engineering techniques in SOPHIE I, II and III. In Sleeman, D. H. and Brown, J. S., editors, *Intelligent Tutoring Systems*, pages 227-282, London: Academic Press.
- Brown, J. S. and Burton, R. B. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, 2:155-192.
- Brown, J. S. and VanLehn, K. (1980). Repair theory: a generative theory of bugs in procedural skills. *Cognitive Science*, 4:479-426.
- Buchanan, B. G. and Feigenbaum, E. A. (1978). Dendral and Meta-Dendral: their application dimensions. *Artificial Intelligence*, 11:5-24.
- Buchanan, B. G. and Mitchell, T. M. (1978). Model-directed learning of production rules. In Waterman, D. A., and Hayes-Roth, F., editors, *Pattern-Directed Inference Systems*, pages 297-312, New York: Academic Press.
- Buchanan, B. G., Mitchell, T. M., Smith, R. G., and Johnson, C. R. (1978). Models of learning systems. In Belzer, J., editor, *Encyclopedia of Computer Science and Technology*, chapter 11, New York: Marcel Dekker. Also Stanford Report STAN-CS-79-692.
- Buchanan, B. G. and Shortliffe, E. H. (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, Mass.: Addison-Wesley.

- Burks, A. W. (1977). *Cause, Chance, Reason: An Inquiry into the Nature of Scientific Evidence*. Chicago: University of Chicago Press.
- Burton, R. and Brown, J. S. (1982). An investigation of computer coaching for informal learning activities. In Sleeman, D. H. and Brown, J. S., editors, *Intelligent Tutoring Systems*, pages 79-98, London: Academic Press.
- Charniak, E. (1977). Ms. Malaprop, a language comprehension system. In *Proceedings of the 1977 IJCAI*, pages 1-7, Cambridge, MA.
- Clancey, W. J. (1979). *Transfer of Rule-Based Expertise Through a Tutorial Dialogue*. PhD thesis, Stanford University. Stanford Technical Report STAN-CS-79-769.
- Clancey, W. J. (1984). NEOMYCIN: reconfiguring a rule-based system with application to teaching. In Clancey, W. J. and Shortliffe, E. H., editors, *Readings in Medical Artificial Intelligence*, chapter 15, pages 361-381, Reading, Mass.: Addison-Wesley.
- Clancey, W. J. (1985). Heuristic classification. *Artificial Intelligence*, 27:289-350.
- Clancey, W. J. (1986). From GUIDON to NEOMYCIN to HERACLES in twenty short lessons. *AI Magazine*, 7:40-60.
- Clancey, W. J. (1987). The knowledge engineer as student: metacognitive bases for asking good questions. In Lesgold, A. and Mandl, H., editors, *Learning Issues for Intelligent Tutoring Systems*, Springer Verlag. In press.
- Clancey, W. J. and Bock, C. (1986). Representing control knowledge as abstract tasks and metarules. In Coombs, M. and Bolc, L., editors, *Computer Expert Systems*, Springer Verlag. Also, Knowledge Systems Lab Report KSL-85-16, Stanford University, April 1985.
- Collins, A., Brown, J. S., and Newman, S. E. (1987). Cognitive apprenticeship: teaching the craft of reading, writing and mathematics. In Resnick, L. B., editor, *Cognition and Instruction: Issues and Agendas*, Lawrence Erlbaum Associates. In press.
- Davis, R. (1982). Application of meta level knowledge in the construction, maintenance and use of large knowledge bases. In Davis, R. and Lenat, D. B., editors, *Knowledge-Based Systems in Artificial Intelligence*, pages 229-490, New York: McGraw-Hill.
- Davis, R. (1984). Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24(1-3):347-410.
- DeJong, G. (1986). An approach to learning from observation. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning, Volume II*, chapter 19, pages 571-590, Los Altos: Morgan Kaufmann.
- DeJong, G. and Mooney, R. (1986). Explanation-based learning: an alternative view. *Machine Learning*, 1(2):145-177.
- DeKleer, J. (1986). An assumption-based TMS. *Artificial Intelligence*, 28(2):127-162.
- DeKleer, J. and Brown, J. S. (1984). A qualitative physics based on confluences. *Artificial Intelligence*, 24(1-3):7-84.

- DeKleer, J. and Williams, B. C. (1987). Diagnosing multiple faults. *Artificial Intelligence*, 32(1).
- Dietterich, T. G. (1984). *Constraint Propagation Techniques for Theory-Driven Data Interpretations*. PhD thesis, Stanford University, Stanford, CA. Stanford Technical Report STAN-CS-84-1030.
- Dietterich, T. G. and Buchanan, B. G. (1981). *The Role of the Critic in Learning Systems*. Heuristic Programming Project Report HPP-81-19, Stanford University, Stanford, CA. Reprinted in Selfridge, O., Rissland, E. and Arbib, M., eds., *Adaptive control of Ill-defined systems*.
- Dyer, M. G. (1973). *In-Depth Understanding*. Cambridge: MIT Press.
- Eshelman, L. and McDermott, J. (1986). MOLE: a knowledge acquisition tool that uses its head. In *Proceedings of the 1986 National Conference on Artificial Intelligence*, Philadelphia, PA.
- Fu, L. and Buchanan, B. G. (1985). *Inductive knowledge acquisition for rule based expert systems*. Knowledge Systems Laboratory Report KSL-85-42, Stanford University, Stanford, CA.
- Genesereth, M. R. (1978). *Automated Consultation for Complex Computer Systems*. PhD thesis, Harvard University.
- Genesereth, M. R. (1981). The role of plans in intelligent teaching systems. In Sleeman, D. H. and Brown, J. S., editors, *Intelligent Tutoring Systems*, pages 137-157, London: Academic Press.
- Genesereth, M. R. (1982). Diagnosis using hierarchical design models. In *Proceedings of the 1982 National Conference on Artificial Intelligence*, pages 278-283, Pittsburgh, PA.
- Genesereth, M. R. (1984). The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24(1-3):411-436.
- Ginsberg, A. (1986). A metalinguistic approach to the construction of knowledge base refinement systems. In *Proceedings of the 1986 National Conference on Artificial Intelligence*, pages 436-441, Philadelphia, PA.
- Ginsberg, A., Weiss, S., and Politakis, P. (1985). SEEK2: a generalized approach to automatic knowledge base refinement. In *Proceedings of the 1985 IJCAI*, pages 367-374, Los Angeles, CA.
- Goldstein, I. (1978). *Developing a Computational Representation for Problem Solving Skills*. Artificial Intelligence Laboratory Memo 495, MIT, Cambridge, MA.
- Gordon, J. and Shortliffe, E. H. (1985). A method for managing evidential reasoning in a hierarchical hypothesis space. *Artificial Intelligence*, 26(3):323-358.
- Hayes-Roth, F., Klahr, P., and Mostow, D. J. (1980). *Knowledge acquisition, knowledge programming, and knowledge refinement*. Technical Report R-2540-NSF, The Rand Corporation, Santa Monica, CA.

- Heckerman, D. (1986). Probabilistic interpretations for Mycin's certainty factors. In Kanal, L. and Lemmar, J., editors, *Uncertainty in Artificial Intelligence*, pages 167-196, New York: North Holland.
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press.
- Holland, J. H. (1986). Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning, Volume II*, chapter 20, pages 593-624, Los Altos: Morgan Kaufmann.
- Johnson, W. L. and Soloway, E. (1985). PROUST: knowledge-based program understanding. *IEEE Trans. Software Engineering*, SE-11(3):267-274.
- Kahn, G., Nowlan, S., and McDermott, J. (1985). MORE: an intelligent knowledge acquisition tool. In *Proceedings of the 1985 IJCAI*, pages 573-580, Los Angeles, CA.
- Korf, R. (1985). *Learning to Solve Problems by Searching for Macro-Operators*. Marshfield, Mass: Pitman.
- Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). SOAR: an architecture for general intelligence. *Artificial Intelligence*, 33(1):1-64.
- Laird, J. E., Rosenbloom, P. S., and Newell, A. (1984). Towards chunking as a general learning mechanism. In *Proceedings of the 1984 National Conference on Artificial Intelligence*, Austin, TX.
- Langley, P. W., Ohlsson, S., and Sage, S. (1984). *A Machine Learning Approach to Student Modeling*. Technical Report RI-TR-84-7, Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA.
- Lenat, D. B. (1976). *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search*. PhD thesis, Stanford University.
- Levesque, H. J. (1984). Foundations of a functional approach to knowledge representation. *Artificial Intelligence*, 23(2):155-212.
- Lindsay, R., Buchanan, B. G., A., F. E., and Lederberg, J. (1980). *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project*. New York: McGraw-Hill.
- London, B. and Clancey, W. J. (1982). Plan recognition strategies in student modeling: prediction and description. In *Proceedings of the 1982 National Conference on Artificial Intelligence*, pages 335-338, Pittsburgh, PA.
- Manna, Z. and Waldinger, R. (1977). *Studies in Automatic Programming Logic*. New York: North-Holland.
- Manna, Z. and Waldinger, R. J. (1975). Knowledge and reasoning in program synthesis. *Artificial Intelligence*, 6(2).

- Michalski, R. S. (1983). A theory and methodology of inductive inference. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An Artificial Intelligence Approach*, chapter 4, pages 83-134, Palo Alto: Tioga Press.
- Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors (1983). *Machine Learning: An Artificial Intelligence Approach*. Palo Alto: Tioga Press.
- Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors (1986). *Machine Learning: An Artificial Intelligence Approach*. Volume II, Los Altos: Morgan Kaufmann.
- Michalski, R. S. and Chilausky, R. L. (1980). Knowledge acquisition by encoding expert rules versus computer by induction from examples: a case study involving soybean pathology. *Int. J. of Man-Machine Studies*, 12(1):63-87.
- Miller, R. A., Pople, H. E., and Myers, J. D. (1984). INTERNIST-1: an experimental computer-based diagnostic consultant for general internal medicine. In Clancey, W. J. and Shortliffe, E. H., editors, *Readings in Medical Artificial Intelligence*, chapter 8, pages 190-209, Reading, Mass.: Addison-Wesley.
- Minton, S. (1985). Selectively generalizing plans for problem solving. In *Proceedings of the 1985 IJCAI*, pages 596-599, Los Angeles, CA.
- Mitchell, T. (1983). Learning and problem solving strategies. In *Proceedings of the 1983 IJCAI*, pages 1139-1151, Karlsruhe, West Germany.
- Mitchell, T., Utgoff, P. E., and Banerji, R. S. (1983). Learning by experimentation: acquiring and refining problem-solving heuristics. In Michalski, T. M., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An Artificial Intelligence Approach*, pages 163-190, Palo Alto: Tioga Press.
- Mitchell, T. M., Carbonell, J. G., and Michalski, R. S., editors (1986). *Machine Learning: A Guide to Current Research*. Boston: Kluwer Academic Publishers.
- Mitchell, T. M., Mahadevan, S., and Steinberg, L. I. (1985). LEAP: a learning apprentice for VLSI design. In *Proceedings of the 1985 IJCAI*, pages 573-580, Los Angeles, CA.
- Newell, A. and Simon, H. A. (1972). *Human Problem Solving*. Engelwood Cliffs: Prentice-Hall.
- Ohlsson, S. and Langley, P. (1985). *Identifying Solution Paths in Cognitive Diagnosis*. Technical Report CMU-RI-TR-85-2, Carnegie-Mellon University, Pittsburgh, PA.
- Patil, R., Szolovits, R., and Schwartz, W. (1981). Causal understanding of patient illness in medical diagnosis. In *Proceedings of the 1981 IJCAI*, pages 893-899, VanCouver, Canada.
- Patil, R. S., Szolovits, P., and Schwartz, W. B. (1982). Information acquisition in diagnosis. In *Proceedings of the 1982 National Conference on Artificial Intelligence*, pages 345-348, Pittsburgh, PA.
- Politakis, P. and Weiss, S. M. (1984). Using empirical analysis to refine expert system knowledge bases. *Artificial Intelligence*, 22(1):23-48.

- Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning*, chapter 15, pages 463-482, Palo Alto: Tioga Press.
- Rich, C., Shrobe, H. E., and Waters, R. C. (1979). Overview of the programmer's apprentice. In *Proceedings of the 1979 IJCAI*, pages 827-828, Tokyo, Japan.
- Russell, S. (1985). *The Compleat Guide to MRS*. Knowledge Systems Laboratory Report KSL-85-108, Stanford University, Stanford, CA.
- Russell, S. (1986). *Inductive and Analogical Reasoning*. PhD thesis, Stanford University, Stanford, CA.
- Sacerdoti, E. (1977). *A Structure for Plans and Behavior*. New York: American Elsevier.
- Samuel, A. L. (1963). Some studies in machine learning using the game of checkers. In Feigenbaum, E. and Feldman, D., editors, *Computers and Thought*, New York: McGraw-Hill.
- Schank, R. C. (1982). *Dynamic Memory*. Cambridge: Cambridge University Press.
- Schmidt, C. F., Sridharan, N. S., and Goodson, J. L. (1978). The plan recognition problem: an intersection of psychology and AI. *Artificial Intelligence*, 11(1-2):45-83.
- Segre, A. M. (1979). *Explanation-based learning of generalized robot assembly plans*. PhD thesis, Univ. of Illinois, Urbana-Champaign.
- Shafer, G. A. (1976). *Mathematical Theory of Evidence*. Princeton: Princeton University Press.
- Shapiro, E. H. (1983a). *Algorithmic Program Understanding*. Cambridge: MIT Press.
- Shapiro, E. Y. (1983b). Logic programs with uncertainties: a tool for implementing rule-based systems. In *Proceedings of the 1983 IJCAI*, pages 529-532, Karlsruhe, West Germany.
- Shaw, D., Swartout, W. R., and Green, C. (1975). Inferring LISP programs from examples. In *Proceedings of the 1975 IJCAI*, Tbilisi, Georgia, USSR.
- Shortliffe, E. H. (1976). *Computer-based Medical Consultations: MYCIN*. New York: American Elsevier.
- Shrobe, H. E. (1979). *Reasoning and Logic for Complex Program Understanding*. PhD thesis, MIT, Cambridge, MA.
- Silver, B. (1986). *Meta-Level Inference*. New York: North Holland.
- Sleeman, D. H. and Brown, J. S., editors (1981). *Intelligent Tutoring Systems*. London: Academic Press.
- Smith, R. G., Winston, H. A., Mitchell, T. M., and Buchanan, B. G. (1985). Representation and use of explicit justifications for knowledge base refinement. In *Proceedings of the 1985 IJCAI*, pages 673-680, Los Angeles, CA.

- Summers, P. (1977). A methodology for LISP program construction from examples. *Journal of the ACM*, 24.
- Sussman, G. J. (1976). *A Computational Model of Skill Acquisition*. New York: Springer-Verlag.
- Swartout, W. R. (1983). XPLAIN: a system for creating and explaining expert consulting programs. *Artificial Intelligence*, 21(3):285-325.
- Thompson, T. and Clancey, W. J. (1986). A qualitative modeling shell for process diagnosis. *IEEE Software*, 3(2):6-15.
- Utgoff, P. E. (1986). *Machine Learning of Inductive Bias*. Boston: Kluwer Academic Publishers.
- VanLehn, K. (1983). *Felicity Conditions for Human Skill Acquisition: Validating an AI-based Theory*. PhD thesis, MIT. Also, Technical Report CIS-21, Xerox PARC, Palo Alto, 1983.
- VanMelle, W. (1980). *A Domain-Independent System that Aids in Constructing Knowledge-Based Consultation Programs*. Heuristic Programming Project Report HPP-80-22, Stanford University, Stanford, CA.
- Waterman, D. (1970). Generalization learning techniques for automating the learning of heuristics. *Artificial Intelligence*, 1:121-170.
- Waterman, D. (1978). Exemplary programming in RITA. In Waterman, D. A. and Hayes-Roth, F., editors, *Pattern-Directed Inference Systems*, New York: Academic Press.
- Wilkins, D. C. (1986). Knowledge base debugging using apprenticeship learning techniques. In *Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 40.0-40.14, Banff, Canada.
- Wilkins, D. C. (1987). Cognitive diagnosis of heuristic classification problem solving. In *Third International Conference on Artificial Intelligence and Education*, page 57, Pittsburgh, PA.
- Wilkins, D. C. and Buchanan, B. G. (1986). On debugging rule sets when reasoning under uncertainty. In *Proceedings of the 1986 National Conference on Artificial Intelligence*, pages 448-454, Philadelphia, PA.
- Wilkins, D. C., Clancey, W. J., and Buchanan, B. G. (1986). An overview of the ODYSSEUS learning apprentice. In Mitchell, T. M., Michalski, R. S., and Carbonell, J. G., editors, *Machine Learning: A Guide to Current Research*, pages 332-340, Boston: Kluwer Academic Publishers.
- Wilkins, D. C., Clancey, W. J., and Buchanan, B. G. (1987a). Using and evaluating differential modeling in intelligent tutoring and apprentice learning system. In Psotka, J. and Massey, D., editors, *Intelligent Tutoring Systems: Lessons Learned*, Hillsdale: Lawrence Erlbaum Associates. In press. Also Stanford Technical Report STAN-CSD-87-1175, Stanford University, Stanford, CA.

- Wilkins, D. C., Clancey, W. J., and Buchanan, B. G. (1987b). Knowledge base refinement by monitoring abstract control knowledge. In Boose, J. and Gaines, B., editors, *Knowledge Acquisition for Knowledge Based Systems*, Academic Press. In press. Also Stanford Technical Report STAN-CSD-87-1182, Stanford University, Stanford, CA.
- Yu. V. L., Fagan, L. M., Wraith, S. M., and Clancey, W. J. (1979). Evaluating the performance of a computer-based consultant. *J. Amer. Med. Assoc.*, 242(12):1279-1282.
- Zadeh, L. A. (1979). Approximate reasoning based on fuzzy logic. In *Proceedings of the 1979 IJCAI*, pages 1004-1010, Tokyo, Japan.

**INDEX**

## INDEX

- "\$ symbol, 19
- action, 7
- bipartite graph minimization
  - formulation, 80
- data request, 7
- detecting deficiencies, 32
- EBG or explanation based
  - generalization, 25
- EBL or explanation based learning"/, 25
- EDB or extensional data base , 16
- essential knowledge, 95
- evaluating repairs, 55
- false negative, 74
- false positive, 74
- genetic algorithm, 35
- global credit assignment, 7
- goal, 38
- heuristic classification, 15
- IDB or intensional data base, 18
- inherent limits of learning, 70
- intelligent editor, 8
- learning by watching a student, 4
- learning by watching an expert system, 4
- learning by watching an expert, 3
- learning by watching, 4
- learning critic*, 7
- learning from experience, 4
- local credit assignment, 7
- LORE or line of reasoning
  - explanation, 38
- lower limits of learning, 101
- metarule, 20
- observable knowledge, 95
- performance element, 6
- performance model of learning, 6
- performance standard, 32
- poison rule, 83
- program, AM, 37
- program, ARMS, 35
- program, DENDRAL, 33,124
- program, EMYCIN, 125
- program, GENESIS, 34
- program, GUIDON-DEBUG, 28
- program, GUIDON2, 125
- program, GUIDON, 125
- program, HACKER, 35
- program, HERACLES, 5,15,121,125
- program, ID3, 22
- program, IMAGE, 29
- program, INDUCE, 22
- program, LEAP, 23
- program, LEX, 36
- program, META-DENDRAL, 33,124
- program, MORE, 23
- program, MRS , 17
- program, MYCIN, 125
- program, NEOMYCIN, 5,125
- program, ODYSSEUS, 119,125
- program, SEEK2, 22
- program, SIERRA, 47
- program, SOAR, 36
- program, TEIRESIAS, 22,126
- referenced knowledge, 95
- relation instances, 17
- relation schema, 16
- relations, 16

sociopathic knowledge base, 71  
sociopathic reduction algorithm, 82  
suggesting repairs, 45  
synthetic agent method, 95  
synthetic agent, 89  
synthetic expert, 95  
synthetic novice, 95  
task interpreter, 20  
task, 20  
training instances, 7,73  
true negative, 74  
true positive, 74  
tuple, 17  
upper limits of learning, 89