

Using Explanation-Based and Empirical Methods in Theory Revision

Dirk Ourston

August 1991 AI 91-164

Abstract:

The knowledge acquisition problem is a continuing problem in expert system development. The knowledge base (domain theory) initially formulated by the expert is usually only an approximation to the correct theory for the application domain. This initial knowledge base must be refined (usually manually) as problems are discovered. This research addresses the knowledge base refinement problem for classification tasks. The research provides an automatic method for correcting a domain theory in the light of incorrect performance on a set of training examples. The method uses attempted explanations to focus the correction on the failing part of the knowledge base. It then uses induction to supply a correction to the knowledge base that will render it consistent with the training examples.

Using this technique, it is possible to correct overly general and overly specific theories, theories with multiple faults at various levels in the theory hierarchy, and theories involving multiple concepts. Methods have been developed for making corrections even in the presence of noisy data. Theoretical justification for the method is given in the form of convergence results that predict that the method will eventually converge to a hypothesis that is within a small error of the correct hypothesis, given sufficient examples. Because the technique currently relies on theorem proving for much of the analysis, it is quite expensive to computationally and heuristic methods for reducing the computational burden have been implemented.

The system developed as part of the research is called EITHER (Explanation-based Inductive Theory Extension and Revision). EITHER uses propositional Horn clause logic as its knowledge representation, with examples expressed as attribute-value lists. The system has been tested in a variety of domains including revising a theory for the identification of promoters in DNA sequences and a theory for soybean disease diagnosis, where it has been shown to outperform a purely inductive approach.

The above-described technical report is available in Taylor 4.140 for \$15.00 per copy from AI Lab Publications - Marlene Bateman (512-471-9567).

**USING EXPLANATION-BASED AND
EMPIRICAL METHODS IN THEORY
REVISION**

DIRK OURSTON

AUGUST 1991 AI 91-164

**USING EXPLANATION-BASED AND EMPIRICAL
METHODS IN THEORY REVISION**

by

DIRK OURSTON, B.S., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCES

THE UNIVERSITY OF TEXAS AT AUSTIN
August, 1991

Acknowledgments

First and foremost, I would like to thank my advisor, Dr. Raymond Mooney. Ray has supported me intellectually, financially, and inspirationally, helping me to overcome many obstacles and to see the substance behind the confusion. Ray was also responsible for the inductive and deductive systems that formed the basis for those components in the project, as well as considerable test software. I would also like to thank the members of my committee, Dr. Ben Kuipers, Dr. Bruce Porter, Dr. Robert Simmons, and Dr. Andrew Whinston, for their interest in and support for my research.

Jeff Mahoney deserves thanks for reviewing an early version of the dissertation as well as straightening out the soybean theory and implementing the fuzzy tester. Hwee Tou Ng provided the abduction system that formed the basis for the abductive component in the project. Shiow-Yang Wu worked extensively on the IOU experiments, which exposed many problems that were later solved by EITHER. The members of the KBANN project at the University of Wisconsin (Jude Shavlik, Geoff Towell and Michael Noordewier) provided not only the DNA theory that was used in the experiments, but helped by providing insight into the DNA theory and the nature of the required theory revisions.

I would also like to thank my brother Leif, who, although a traffic engineer, painstakingly reviewed an early copy of the dissertation and exposed many style weaknesses, in particular my love of, commas. Finally, thanks to the explanation group, for many interesting and heated discussions particularly when Brad Blumenthal was still around, that helped me to put my own work in perspective.

*The University of Texas at Austin
August, 1991*

Dirk Ourston

USING EXPLANATION-BASED AND EMPIRICAL METHODS IN THEORY REVISION

Publication No. _____

Dirk Ourston, Ph.D.
The University of Texas at Austin, 1991

Supervising Professor: Raymond J. Mooney

The knowledge acquisition problem is a continuing problem in expert system development. The knowledge base (domain theory) initially formulated by the expert is usually only an approximation to the correct theory for the application domain. This initial knowledge base must be refined (usually manually) as problems are discovered. This research addresses the knowledge base refinement problem for classification tasks. The research provides an automatic method for correcting a domain theory in the light of incorrect performance on a set of training examples. The method uses attempted explanations to focus the correction on the failing part of the knowledge base. It then uses induction to supply a correction to the knowledge base that will render it consistent with the training examples.

Using this technique, it is possible to correct overly general and overly specific theories, theories with multiple faults at various levels in the theory hierarchy, and theories involving multiple concepts. Methods have been developed for making corrections even in the presence of noisy data. Theoretical justification for the method is given in the form of convergence results that predict that the method will eventually converge to a hypothesis that is within a small error of the correct hypothesis, given sufficient examples. Because the technique currently relies on theorem proving for much of the analysis, it is quite expensive computationally and heuristic methods for reducing the computational burden have been implemented.

The system developed as part of the research is called EITHER (Explanation-based Inductive Theory Extension and Revision). EITHER uses propositional Horn clause logic as its knowledge representation, with examples expressed as attribute-value lists. The system has been tested in a variety of domains including revising a theory for the identification of promoters in DNA sequences and a theory for soybean disease diagnosis, where it has been shown to outperform a purely inductive approach.

Table of Contents

Acknowledgments	iv
Abstract	v
Table of Contents	vi
List of Tables	x
List of Figures	xi
1. Introduction	1
1.1 Background	1
1.1.1 EBL	2
1.1.2 SBL	3
1.1.3 Integrated Approaches	3
1.1.4 EITHER	4
1.2 Organization of the Thesis	8
2. Single Category Theories	10
2.1 EITHER's Response to Theory Errors	10
2.2 An Example Theory	10
2.3 Finding the Minimum Covers	12
2.3.1 The Minimum Antecedent Cover	13
2.3.2 The Minimum Rule Cover	17
2.4 Theory Generalization	20
2.4.1 Antecedent Retraction	21

2.4.2	Antecedent Generalization	22
2.4.3	Inductive Rule Generalization	25
2.5	Theory Specialization	25
2.5.1	Rule Retraction	27
2.5.2	One-Sided Specialization	28
2.5.3	Inductive Rule Specialization	29
2.6	Experimental Results	30
3.	Revising Intermediate Rules	34
3.1	Consequent Identification	34
3.2	Antecedent Construction	40
3.2.1	Concept Utilization	42
3.2.2	Rule Reduction	43
4.	Multiple Category Theories and the Correctability Problem	47
4.1	The Reasons for the Correctability Problem	49
4.2	Theory-based Response	50
4.3	Cover-based Response	51
4.4	Implementation Simplifications	55
4.5	Experimental Results	55
5.	Improving Shared Rules	58
5.1	Analysis	58
5.2	Experimental Results	60
6.	Response to Noise	61
6.1	Accounting for Noise in the Minimum Cover	61

6.2	Accounting for Noise in One-Sided Corrections	65
6.3	Accounting for Noise in Inductive Rule Formation	66
6.4	Noise Performance Results	67
7.	Convergence Results	75
7.1	Learnability	75
7.2	An Argument for Consistency	77
7.2.1	Consistency of the Generalization Algorithm	77
7.2.2	Consistency of the Specialization Algorithm	79
8.	Computational Complexity	82
8.1	Analysis	82
8.2	Experimental Results	85
9.	Related Work	87
9.1	RTLS	88
9.2	ML-SMART	89
9.3	FOCL	89
9.4	KBANN	90
9.5	IOE	90
9.6	ANA-EBL	90
9.7	IOU	91
9.8	LISE	91
9.9	CIGOL	91
9.10	Gemini	92
9.11	OCCAM	92
9.12	ODYSSEUS	93

9.13 COAST	94
9.14 KI	94
10.Future Work	95
10.1 Improving the Knowledge Representation	95
10.2 Efficiency Improvements	96
11.Conclusions	98
A. The EITHER algorithm	100
B. The Minimum Antecedent Cover Algorithm	109
C. The Minimum Rule Cover Algorithm	112
C.1 The Greedy Rule Cover Algorithm	112
C.2 Exhaustive Rule Cover Algorithm	113
D. Domain Theories for the Soybean and DNA Theories	115
D.1 Soybean Theory	115
D.2 DNA Theory	118
E. Detailed Correction Results	119
E.1 DNA Theory	119
E.2 Soybean Theory	122
BIBLIOGRAPHY	136
Vita	

List of Tables

8.1 Complexity Results	84
------------------------------	----

List of Figures

1.1 EITHER Architecture	5
1.2 Theory Error Taxonomy	6
1.3 EITHER Architecture	7
2.1 EITHER System Response to Theory Errors	11
2.2 The Cup Theory	11
2.3 Cup Examples	12
2.4 Theory Tree for the Cup Theory	14
2.5 EITHER Trace for the Minimum Antecedent Cover Example	16
2.6 EITHER Generalization Response to Theory Errors	21
2.7 Antecedent Popup Example	22
2.8 The Cup Theory With Two Missing Rules.	23
2.9 Generalizing an Existing Antecedent	24
2.10 The EITHER Response to an Additional Antecedent	24
2.11 Missing Rule Example	26
2.12 EITHER Specialization Response to Theory Errors	26
2.13 EITHER Response to Additional Rule	28
2.14 Specializing a Linear Antecedent	29
2.15 EITHER Inductive Rule Specialization	30
2.16 EITHER Results for the DNA Theory	31
3.1 The Complete Animal Theory	36
3.2 Modified Animal Theory Segment	39
3.3 Different Types of Gaps in Incomplete Theories	41
3.4 The Cup Theory	41
3.5 Using Intermediate Concepts	43
3.6 Reducing Rules	46
4.1 A Simple Inconsistent Theory	49
4.2 EITHER Performance Results for the Soybean Theory	57
5.1 Animal Domain Theory	62
5.2 Computer Domain Theory	63

5.3	Corrupted Animal Theory	64
5.4	Corrupted Computer Theory	64
5.5	Cross-category Learning in the Animal and Computer Domains	69
5.6	Standard Learning in Animal and Computer Domains	70
6.1	Test Accuracy vs. Noise Level for Animal and Computer Domains	71
6.2	Train Time vs. Noise Level for Animal and Computer Domains	72
6.3	Theory Complexity vs. Noise Level for Animal and Computer Domains	73
6.4	Test Accuracy vs. Noise Level for DNA Domain	74
8.1	Run Time versus Training Set Size for the DNA and Soybean Theories	86
9.1	Related Work	87

Chapter 1

Introduction

In general, one of the most difficult problems in expert system development is the construction of the knowledge base. As a result, the rate of progress in developing useful expert systems is directly related to the speed with which expert knowledge bases can be assembled. Many of the developments in machine learning have been attempts to solve the *knowledge acquisition problem*, whereby an expert's domain knowledge is summarized in an efficient formal representation.

The knowledge acquisition problem can be divided into two phases: an initial phase, in which a knowledge engineer extracts initial rough knowledge from an expert, and the *knowledge base refinement phase*¹, in which the initial knowledge base is refined into a high performance knowledge base [Ginsberg, 1988]. The initial knowledge base is acquired as whole rules, or whole sets of rules, that are used to represent various concepts in the domain. In contrast, during the knowledge base refinement phase, *components* of the existing rules are modified, in addition to adding and deleting rules, in an effort to improve the *empirical adequacy* of the knowledge base, that is, its ability to reach correct conclusions within its problem space.

This dissertation presents a method for automating the knowledge base refinement process, as it applies to classification systems expressed in propositional Horn Clause logic. The method assumes that an approximately correct knowledge base has been provided, from textbook knowledge or rough knowledge from an expert. The knowledge base represents a theory for the domain that the expert system addresses. The method identifies corrections to the knowledge base that are not only syntactically and semantically admissible, but which also preserve, as far as possible, the expert's initial version of the rules. Within this context, the final purpose of the corrections is to improve the empirical adequacy of the knowledge base.

1.1 Background

One way of solving the knowledge acquisition problem is to use machine learning techniques. In recent years, two major approaches to machine learning have emerged. These are: *explanation-based learning* (EBL) [Mitchell *et al.*, 1986], where generalizations are made from single examples which satisfy the constraints of a given domain theory, and *similarity-based learning* (SBL) [Dietterich and Michalski, 1983], which inductively generalizes from a series of examples in a way that is consistent with the examples. Later research has attempted to merge explanation-based and similarity-based methods to improve the performance of both. This section provides a brief review of some of the techniques that have been used

¹[Bareiss *et al.*, 1989] divide the knowledge base refinement stage into two phases: the first, which establishes the correctness of the knowledge base, and a second which is concerned with efficiency.

in each area, and concludes with a description of the EITHER (Explanation-based Inductive Theory Extension and Revision) system, that was developed in support of this research.

1.1.1 EBL

In Explanation-based learning, a theory is used to solve a problem using facts from an example. The types of problems solved by EBL include: proving that the example is a member of a concept; predicting the outcome of an event represented by the example; and using the example in the context of some problem solving activity, such as scheduling. The facts pertaining to the example are then generalized using the theory to find the *weakest preconditions* necessary for the problem solution.

A theory for use in explanation-based learning may be specified in many different ways, as long as the theory is used in the solution of problems relevant to the domain. For example, a *procedural* theory is one which is specified operationally, so that the problem solution is accomplished through a series of operational steps which lead to the desired goal. When the theory is procedural, the generalization is accomplished by determining the most general conditions under which the sequence of actions will lead to the goal. This approach is used, for example, in the STRIPS system [Fikes *et al.*, 1972].

In other cases the problem formalism may consist of schemas connected by semantic links of various kinds. When the theory is expressed by semantically-linked schemas, the generalization consists of abstractions to the schemas used in the problem solution. An example of this approach is found in the OCCAM system [Pazzani, 1988b].

In the case of deductive solutions to problems, the specification language will in general be some form of logic such as predicate calculus or propositional logic. With deductive problem solving, the generalization consists of removing constraints from the definition of the example which were not used in its proof, and expressing it in terms of operational predicates (predicates from the theory which are present in the example or which represent facets of the problem description which are directly observable) [Mitchell *et al.*, 1986]. This approach is used in the EGGS system [Mooney, 1988].

One of the limitations of explanation-based learning is that this technique is only capable of learning concepts which are in the deductive closure of the original theory. The assumption is that the sub-concepts so learned are useful ones which will contribute to the computational efficiency with which future examples are processed.

1.1.2 SBL

Similarity-based learning, on the other hand, makes no use of an initial theory. There are many different forms of similarity-based or empirical learning, but one thing they all have in common is that a data structure is created which is a generalization of the positive examples and which accounts for none of the negative examples (if any are provided). Examples of such data structures include decision trees [Quinlan, 1986a], which use the *discrimination net* approach to classifying examples, and version spaces [Mitchell, 1978], which represent a hypothesis space of possible concepts which are consistent with the provided examples.

Decision trees are constructed for feature-valued examples using an information-theoretic approach to first select the most useful feature for discriminating between positive and negative examples. For each value of this feature the next most useful feature for discriminating examples which have the given feature value is chosen. This process continues, using features in descending order of importance, until a feature value is reached which separates the remaining examples into a single class.

In the case of version spaces, the hypotheses are represented as two sets of objects: the *maximally general specializations* (MGSs) and the *maximally specific generalizations* (MSGs). MGSs and MSGs are roughly akin to the least upper bound and the greatest lower bound in mathematics. For each new example, if it is a positive example, each MSG is generalized so the examples it represents include the new example and do not include any of the negative examples. If the new example is a negative example, each MGS is specialized so as to not include the example and still include all of the positive examples. When the MGSs and MSGs converge, the learning process is complete.

Because the concept definitions in similarity-based learning are inductively acquired, in the case of complicated theories many thousands of examples may be required to acquire the theory.

1.1.3 Integrated Approaches

Because of the limitations noted above, very recent work in machine learning has focused on ways to integrate explanation-based and similarity-based learning to capture the best features of both and eliminate the weaknesses. The techniques include using SBL to focus EBL, using SBL to acquire knowledge for EBL, and using EBL to focus SBL.

The UNIMEM system [Lebowitz, 1986b] is an example of using SBL to focus EBL. This system has been tested in the realm of congressional voting records. When it detects similarities in the backgrounds of congressmen who voted for or against a particular bill, it tries to explain how the similar features could account for the voting position. It then uses the explanation to identify the causally relevant features among the example data. Lebowitz claims that this technique helps to focus the explanation process and make it more efficient while eliminating spurious correlations to which a purely SBL system is susceptible.

Using SBL to acquire knowledge for EBL is demonstrated by the OCCAM system [Pazzani, 1987]. OCCAM starts with a primitive causal theory which includes concepts such as: if one event immediately precedes another in time and is located in physical proximity, then it can be induced that the first event "causes" the second. Using this primitive theory, OCCAM is capable of discovering rules such as: family members have the goal of protecting the well being of other family members. These secondary rules are then used to construct explanations for new events which are presented to the system. In this way the system is able to inductively acquire the theory that is later used by the explanation-based portion of the system.

An example of using EBL to focus SBL is the LEX2 system [Utgoff, 1986]. LEX2 analyzes the solutions to calculus integration problems using explanations to determine the features of the problem which enabled a particular operator to be used in the solution. The analytically generalized examples containing only important features are then combined using SBL to determine general heuristics for when to apply operators.

1.1.4 EITHER

The EITHER system combines explanation-based and empirical learning to provide a focused correction to an incorrect theory. The explanation-based part of the system is used to identify the failing parts of the theory, and to constrain the examples used for induction. The empirical part of the system determines the specific corrections to failing rules which cause them to be consistent with the supplied examples.

Stated succinctly, the purpose of EITHER is:

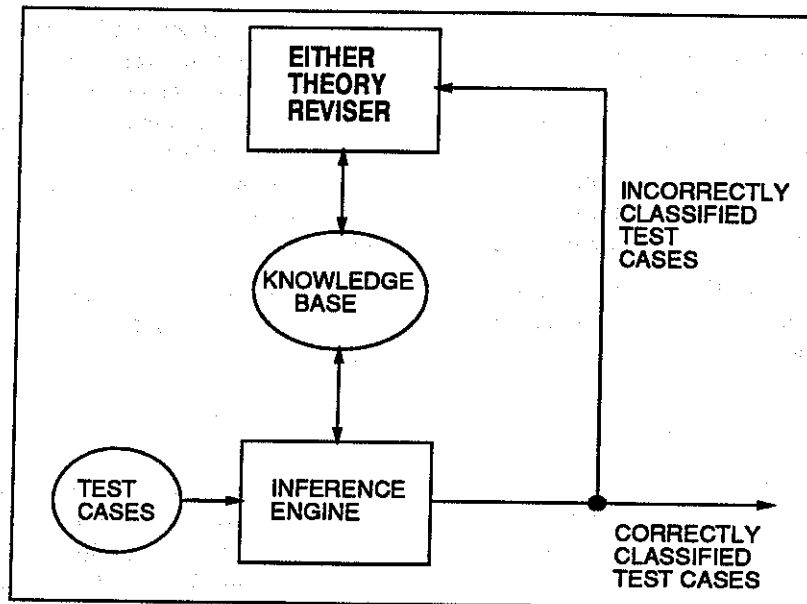


Figure 1.1: EITHER Architecture

Given: an existing domain theory which approximately defines a concept, and a set of positive and negative examples of the concept,

Find: a minimally revised version of the theory which classifies the examples correctly.

It is difficult to precisely define the adjective “minimal” used to characterize the revision to be produced. Since it is assumed that the original theory is “approximately correct” the goal is to change it as little as possible. Syntactic measures such as the total number of symbols added or deleted are reasonable criteria. EITHER uses various methods to help insure that its revisions are minimal in this sense. However, finding a revision that is guaranteed to be syntactically minimal is clearly computationally intractable. When the initial theory is empty, the problem reduces to that of finding a minimal theory for a set of examples.

Figure 1.1 shows the architecture for the EITHER system. So long as the expert system correctly classifies test cases, no additional processing is required. In the event that a misclassified example is detected, EITHER is used to correct the error.

EITHER creates a minimal correction to the theory in the form of modifications to existing rules or the addition or deletion of rules. Since EITHER starts with an approximate theory, fewer examples are required to learn the correct theory, when compared to purely inductive methods. In addition, EITHER improves the comprehensibility of the resulting knowledge since it preserves the structure of the original theory as much as possible. In comparison, inductive approaches learn a theory which is expressed in a logical form which may not include the intermediate concepts used by humans.

Horn-clause logic was chosen as the formalism for the EITHER system. This provides a relatively simple language for exploring the problems associated with inductive theory revision. Since the language is Turing-complete, other formalisms can be equivalently expressed in the logic (for example, schema hierarchies can be expressed as equivalent rule

chains). Theories used by the EITHER system are constrained to be expressed in an extended² version of propositional logic. In addition, the domain theories are required to be acyclic and therefore each theory defines a directed acyclic graph (DAG). For the purpose of theory refinement, EITHER makes a closed-world assumption. If the theory does not prove that an example is a member of a concept, then it is assumed to be a negative example of that concept. The domain theories upon which EITHER has been tested have all corresponded to *classification* tasks - identifying whether a particular example is a member of a concept.

Propositions that are used to describe the examples (e.g. (color black)) are called *observables*. To avoid problems with negation as failure, only observables can appear as negated antecedents in rules. Propositions that represent the final concepts in which examples are to be classified are called *categories*. For example, in a theory for birds the various classes for birds, such as pigeon, duck or grackle would be typical categories. It is currently assumed that the categories are disjoint. In a typical domain theory, all of the sources (leaves) of the DAG are observables and all of the sinks (roots) are categories. Propositions in the theory that are neither observables nor categories are called *intermediate concepts*.

The need for EITHER processing is signalled by incorrectly classified examples, as shown in the Figure 1.1. In making corrections, EITHER operates in batch mode, using as input a set of *training examples*. These training examples would normally correspond to examples that had been incorrectly classified by the initial theory, as well as the remaining correctly classified examples. The incorrectly classified examples, or *failing* examples, are used to identify that there is an error and to control the correction. The correctly classified examples are used to focus the correction and to limit the extent of the correction. An important property of the EITHER algorithm is that it is guaranteed to produce a revised theory that is consistent with the training examples when there is no noise present in the training examples.

Figure 1.2 shows a taxonomy for incorrect propositional theories. At the top level, theories can be incorrect because they are either overly general or overly specific.

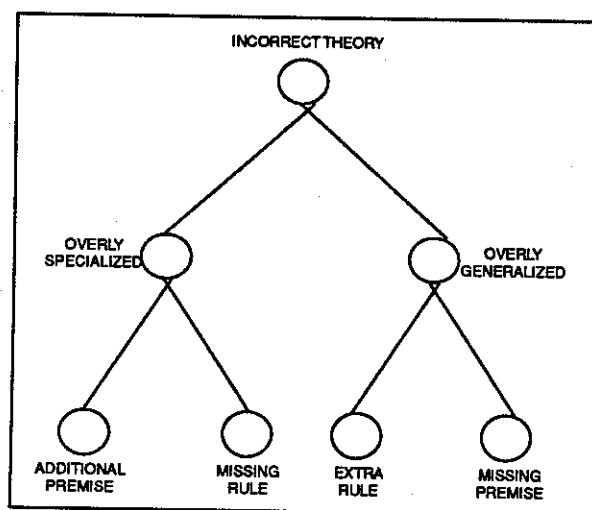


Figure 1.2: Theory Error Taxonomy

²The extension allows specifying numerical intervals and discrete values in the antecedents for rules.

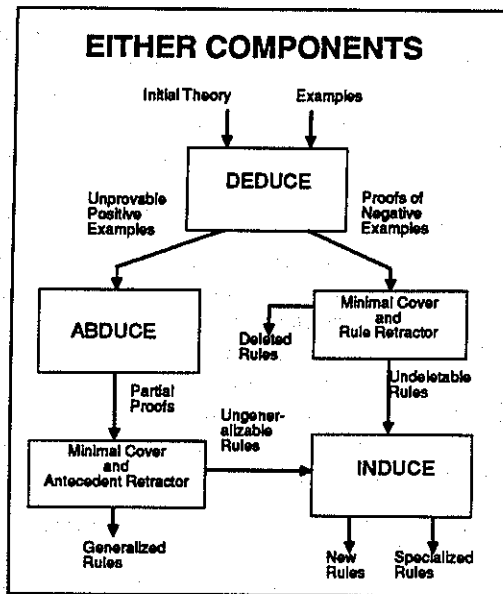


Figure 1.3: EITHER Architecture

An overly general theory contains a concept definition that entails concept membership for examples which are not members of the concept. In general, this will result in negative examples for a category being proven as members of the category. One way an overly general theory can happen is when rules in a concept definition lack required antecedents, providing proofs for examples which should have been excluded. Another way in which examples can be erroneously included is by having additional rules in the concept definition which are not correct. The additional rules provide proofs of concept membership for examples which do not properly belong to the concept.

By contrast, an overly specific theory fails to entail concept membership for members of a concept. This can occur because the theory is missing a rule which is required in the proof of concept membership, or because the existing rules have additional antecedents which exclude concept members.

The following terminology is used in this section, as well as the remainder of the dissertation. "the example is provable," is used to mean "the example is provable as a member of its own category." "a failing positive example" is used to mean "a positive example which is not provable." "a failing negative example" is used to mean "the example is provable as a member of other than its own category."

EITHER uses a combination of methods to achieve the knowledge base revision, as shown in Figure 1.3. EITHER initially uses deduction (DEDUCE) to identify failing examples among the input example set. During the course of the correction, deduction is also used to assess proposed changes to the theory as part of the generalization and specialization processes. For example, if a change is made to the theory to generalize it to cover a failing positive example, deduction would be used to determine whether the change causes negative examples to become provable.

EITHER uses abduction (ABDUCE) to initially find the failing part of an approximate theory for overly specific theories. Most specific abduction [Stickel, 1988] is used to identify the antecedents in the theory which conflict with the facts given in the examples. This is done by relaxing the requirement that all abductive assumptions be consistent. With

this requirement relaxed, ABDUCE is able to identify facts in a failing positive example which, if assumed, would permit a proof of category membership for the example. The next step in the process is to correlate these assumptions with the proof rule in which they are required. The antecedents in the rule which correspond to these assumptions are then termed *conflicting antecedents*, since they conflict with the facts of the example. An important property of these antecedents is that if they are removed from their corresponding rules, the corresponding failing positive examples all become provable. As a result, proofs containing conflicting antecedents are termed *partial proofs*, since the conflicting antecedents would need to be retracted in order for the proof to be complete. The rules containing conflicting antecedents are used to form a *cover* for the failing positive examples. A cover is a set of rules which, if generalized, would cause all of the failing positive examples to be provable.

Induction (INDUCE) is used under certain conditions to correct particular rules based on the failing examples associated with the rule. EITHER uses ID3 [Quinlan, 1986a] as its inductive component. The decision trees returned by ID3 are translated into equivalent Horn-clause rules [Quinlan, 1987]. The remaining components of the EITHER system constitute generalization and specialization control algorithms, which identify and specify the types of corrections to be made to the theory.

One of the main advantages of the EITHER architecture is its modularity. Because the control and processing components are separated from the deductive, inductive, and abductive components, these latter components can be modified or replaced as the need arises. For example, the abduction algorithm currently implemented in ABDUCE is provably exponential in the size of the theory. However, this algorithm can be exchanged for one using ATMS (Assumption-based Truth Maintenance System) processing and beam search ([Ng and Mooney, 1991]) to improve the efficiency of the algorithm, without noticeably affecting the remainder of the EITHER system.

1.2 Organization of the Thesis

The remainder of the thesis is organized into several sections, which discuss the EITHER approach to theory correction. Chapter 2 presents the major components of the EITHER revision algorithm: generating the cover, theory generalization, and theory specialization in the context of a *single category theory*. A single category theory is one in which all of the examples are either positive or negative instances of a single concept, or category.

Chapter 3 presents EITHER's approach to identifying and utilizing intermediate concepts within the theory and recognizing new concepts in the rules which are created. Chapter 3 shows how EITHER can make changes at any level in the input theory. Constructive induction techniques are used for intermediate concept creation and utilization.

Chapter 4 presents the extensions to the EITHER algorithm that are required when multiple category theories are introduced. The main complication introduced by multiple category theories is the *correctability problem*: multiple category theories can contain rules which participate in the mis-classification of examples but which cannot be corrected to solve the problem.

Chapter 5 shows the advantage of EITHER in improving theories which contain shared concepts. With shared concepts EITHER can be used to improve a system's performance in categories which have not been covered by the training examples.

Chapter 6 describes EITHER's response when noise is present in the input examples. In the noise-free case, the corrected theory which EITHER generates is guaranteed to be consistent with the training examples. However, when noise is introduced, this claim can no longer be made. Instead, EITHER invokes a noise response algorithm which attempts

to avoid overfitting the training data. In this case, EITHER should perform at roughly an equivalent level over both the training and test data.

Chapter 7 shows that the revised theory learned by EITHER is consistent with the training examples, in the noise-free case. Because of this result, EITHER can be shown to converge to an approximately correct hypothesis most of the time, given sufficient training examples. Chapter 8 presents an analysis of the complexity of the EITHER algorithm.

Chapter 9 reviews other approaches to the theory revision problem and compares them to the approach used by EITHER. Chapter 10 discusses the open research questions which have been uncovered by this research. Chapter 11 summarizes the results of the research and the conclusions that can be drawn from the presented results.

Finally, the appendices present the details of the EITHER cover, generalization, and specialization algorithms, as well as full presentations of the domain theories used in the experiments and some detailed experimental results.

Chapter 2

Single Category Theories

This chapter presents a simplified version of the EITHER algorithm appropriate to single category learning. Single category learning occurs when both the initial theory and the examples refer to just a single concept, or category. Restricting the scope of the discussion in this way allows the presentation of the essential aspects of the theory revision algorithm, without extraneous details. Extensions to this chapter include: noise in the input examples, Chapter 6; locating and correcting intermediate theory rules, Chapter 3; and the correction of theories representing multiple concepts or categories, Chapter 4.

2.1 EITHER's Response to Theory Errors

Figure 2.1 shows the EITHER response when given an incorrect theory and correctly classified training examples. In the single category case, examples are classified by simply attempting to prove them with the theory. Examples which are provable are labeled as positive and all other examples are labeled as negative. A positive example that is not provable by the theory signals the need for theory generalization, as explained in Section 2.4. When a negative example is erroneously provable by the theory, theory specialization is called for, the subject of Section 2.5. The corrections made by these algorithms are independent: a theory may be generalized, specialized, or both, as dictated by the theory errors. The input to the theory generalization or specialization algorithm is a *cover*, a set of rules requiring correction (see Section 2.3).

2.2 An Example Theory

In explaining the EITHER algorithm, the "Cup Theory," originally developed by Winston [Winston *et al.*, 1983], will be used (see Figure 2.2). In essence, the theory states that something is a cup if it is stable, liftable and an open vessel. Something is stable if it has a bottom, and the bottom is flat. Something is liftable if it is graspable and lightweight. Something is an open vessel if it has a concavity, and the concavity points upward. Something is graspable if either it has a handle, or its width is small and it is insulating. Finally, a material is insulating if it is either styrofoam or ceramic.

Figure 2.3 shows six examples that are consistent with this initial theory. Each example consists of the eight boolean features

**has-concavity, upward-pointing-concavity, has-bottom, flat-bottom, lightweight,
has-handle, styrofoam and ceramic;**

that are as defined for the theory, the three discrete features **color**, **width**, and **shape**; where a discrete feature is a feature having an enumerable set of values, and the single linear feature **volume**; where a linear feature has a numerical value.

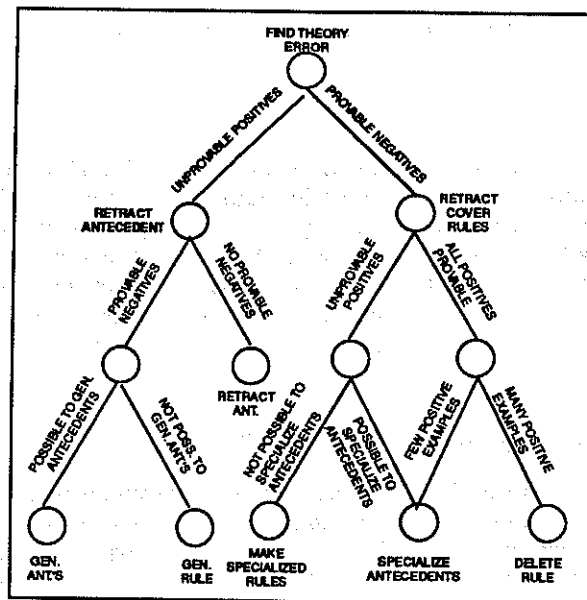


Figure 2.1: EITHER System Response to Theory Errors

(cup)	←	(stable) ∧ (liftable) ∧ (open-vessel)
(stable)	←	(has-bottom) ∧ (flat-bottom)
(liftable)	←	(graspable) ∧ (lightweight)
(graspable)	←	(has-handle)
(graspable)	←	(width small) ∧ (insulating)
(insulating)	←	(material styrofoam)
(insulating)	←	(material ceramic)
(open-vessel)	←	(has-concavity) ∧ (upward-pointing-concavity)

Figure 2.2: The Cup Theory







	has-concavity	upward-pointing	has-bottom	flat-bottom	lightweight	has-handle	styrofoam	ceramic	color	width	volume	shape
1. +	X	X	X	X	X	X		red	sm	8	hem	
2. +	X	X	X	X	X	X		blue	med	16	hem	
3. +	X	X	X	X	X		X	tan	med	8	cyl	
4. -	X	X	X	X	X			gray	sm	8	cyl	
5. -	X	X	X	X	X	X		red	med	8	hem	
6. -	X	X	X	X	X		X	blue	med	16	hem	

Figure 2.3: Cup Examples

The six examples and theory defined above will be used throughout the remainder of this chapter to illustrate various features of the EITHER algorithm.

2.3 Finding the Minimum Covers

A *cover* is a construct that forms the basis for the theory correction algorithm. As used by EITHER a cover is a set of rules requiring correction. There are two types of covers used by the EITHER algorithm: the *antecedent cover* for theory generalization, and the *rule cover* for theory specialization. The antecedent cover is so named because, for each rule in the cover, the *antecedents* requiring correction are identified. In the rule cover the rule itself must be corrected, rather than particular antecedents.

Each type of cover is used by the EITHER algorithm to guarantee the correct classification of one type of example (that is, to cause all unprovable positive examples to be provable, or to cause all negative examples that are provable to no longer be provable). The antecedent cover is used by the generalization algorithm to guarantee that positive examples will be provable¹. The rule cover is used by the specialization algorithm to guarantee that no negative example is provable² by the theory. The unprovable positive examples are therefore associated with the antecedent cover, and the provable negative examples are associated with the rule cover. There is an essential property that holds for both types of cover:

If all of the elements of the cover are removed from the theory, the examples associated with the cover will be correctly classified.

In each case, EITHER selects a *minimum* cover for use in theory correction. The details of the minimum cover algorithms are given in the next two sections.

¹As a member of their concept, in the case of multiple categories, see Chapter 4.

²In *other* than their own category, for the case of multiple categories, see Chapter 4.

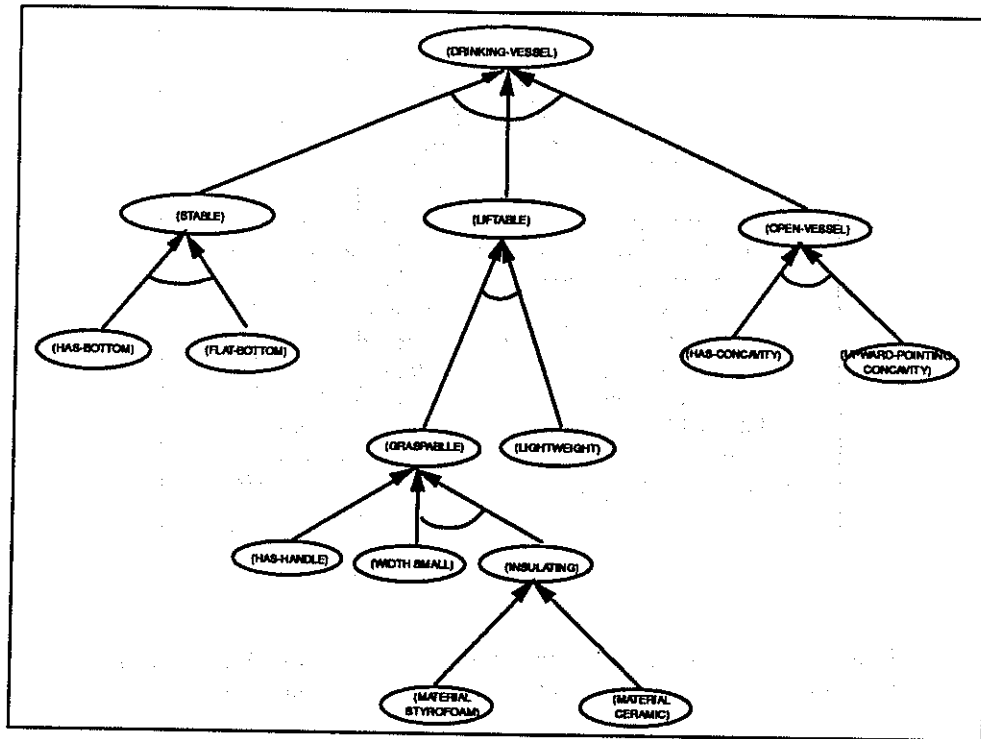


Figure 2.4: Theory Tree for the Cup Theory

2.3.1 The Minimum Antecedent Cover

This section presents an overview of the minimum antecedent cover algorithm, and describes the two versions of the algorithm that have been implemented for the EITHER system. The complete Minimum Antecedent Cover Algorithm is presented in Appendix B

Algorithm overview. The possible proofs of a given goal in a theory can be represented as an and-or tree (or in the more general case an and-or graph), that will be called the *theory tree*. Figure 2.4 shows the theory tree corresponding to the cup theory. This tree may be partitioned into a set of and-trees, one for each possible combination of or-branches in the original theory, and each one representing a separate possible proof. These are traditionally called proof trees.

For each such proof tree the leaves of the tree may or may not unify with facts for a particular example. In the event that they do not unify, the system will identify the antecedents that, if retracted, would permit a complete proof of the given example. As mentioned in Section 1.1.4 these are called *conflicting antecedents*, since they conflict with the facts for the example, and each such proof is deemed a *partial proof* since rule antecedents must be retracted in order for it to be complete.

In a complex system, there will be many such partial proofs for each unprovable example. In order to minimize the changes to the initial theory, the Occam's razor heuristic of finding the minimum number of conflicting antecedents required to cover all of the failing examples has been adopted. This set is the *minimum antecedent cover* that is input to the generalization algorithm.

Stating the problem as a logical expression we have:

$$E_1 \wedge E_2 \wedge \dots \wedge E_n$$

where E_i represents the statement that the i th unprovable positive example has a completed partial proof, that is,

$$E_i \equiv P_{i1} \vee P_{i2} \vee \dots \vee P_{im}$$

where P_{ij} represent the statement that the j th partial proof of the i th unprovable example is completed, that is,

$$P_{jk} \equiv A_{jk1} \wedge A_{jk2} \dots \wedge A_{jkp}$$

where the A_{ijk} means that the k th conflicting antecedent used in the j th partial proof of the i th example is retracted. In order to determine a minimum change to the theory, we need to find the minimum set of conflicting antecedents (A 's) that satisfy this expression.

To illustrate these concepts more concretely, let's assume that the rule

$$(\text{graspable}) \leftarrow (\text{has-handle})$$

is removed from the cup theory. This will cause examples 2 and 3 from Figure 2.3 to no longer be provable in this overly specific version of the cup theory. The "E" expressions from above would be:

1. $E_2 \equiv (\text{width small}) \vee ((\text{width small}) \wedge (\text{material ceramic}))$
2. $E_3 \equiv (\text{width small}) \vee ((\text{width small}) \wedge (\text{material styrofoam}))$

That is, the conflicting antecedents associated with the two partial proofs of example 2 are as given in equation 1, and the conflicting antecedents associated with example 3 are as given in equation 2. Example 1 is not included in the expressions, since it is provable in the initial theory and so would not be input to the minimum antecedent cover algorithm in the first place. The possible covers for the failing examples consist of all combinations of the conflicting antecedents associated with one partial proof of each of the examples. For this illustration the potential covers would be (after logical redundancies have been removed):

1. (width small)
2. ((width small) \wedge (material ceramic))
3. ((width small) \wedge (material styrofoam))
4. ((width small) \wedge (material ceramic) \wedge (material styrofoam))

From these covers EITHER chooses (width small) as the minimum length cover³. Figure 2.5 shows the EITHER trace that corresponds to this illustration.

³Actually, EITHER uses a benefit-to-cost ratio, as defined later in this section, to obtain the antecedent cover. In most cases this will result in the minimum length cover being selected.

Finding minimum antecedent cover.

The potential conflicting antecedents and associated examples are:

antecedents: (width small)(material styrofoam)

associated example:

(cup (has-concavity) (upward-pointing-concavity)
 (has-bottom) (flat-bottom) (lightweight) (has-handle)
 (material ceramic) (color tan) (width medium)
 (volume 8) (shape cylindrical))

antecedent: (width small)

associated examples:

(cup (has-concavity) (upward-pointing-concavity)
 (has-bottom) (flat-bottom) (lightweight) (has-handle)
 (material ceramic) (color tan) (width medium)
 (volume 8) (shape cylindrical))

(cup (has-concavity) (upward-pointing-concavity)
 (has-bottom) (flat-bottom) (lightweight) (has-handle)
 (material styrofoam) (color blue) (width medium)
 (volume 16) (shape hemispherical))

antecedents: (width small)(material ceramic)

associated example:

(cup (has-concavity) (upward-pointing-concavity)
 (has-bottom) (flat-bottom) (lightweight) (has-handle)
 (material styrofoam) (color blue) (width medium)
 (volume 16) (shape hemispherical))

Expanding cover. 2 examples remain to be covered.

Expanding cover. 0 examples remain to be covered.

The selected antecedent cover is:

((graspable) <- (width small) (insulating))

Conflicting antecedent:

((width small))

Antecedent cover complete.

Figure 2.5: EITHER Trace for the Minimum Antecedent Cover Example

Implemented Versions. Two versions of the minimum antecedent cover algorithm have been implemented, one based on the *greedy* algorithm [Johnson, 1974], and the other based on the *branch and bound* algorithm [Lawler and Wood, 1966]. The greedy algorithm does not guarantee to find the minimum cover, but will come within a logarithmic factor of it [Johnson, 1974]. This process operates in time polynomial with the number of examples and linear with respect to the size of the theory. The branch and bound algorithm is the same as the A* algorithm [Nilsson, 1980], with $h = 0$. It is guaranteed to find the minimum length cover that accounts for all of the examples, but the process may take exponential time in the number of examples.

The greedy cover algorithm used by EITHER first forms the set of unique partial proofs associated with the failing positive examples. It then iteratively updates a partial cover for the examples, as follows. At each iteration, the algorithm chooses a partial proof and adds the conflicting antecedents associated with the proof to the partially formed minimum cover from the previous cycle. The selection criteria for determining which partial proof to add is the *benefit-to-cost* associated with the partial proof, where the benefit-to-cost is defined as the ratio of the additional examples covered when the conflicting antecedents are added to the partial cover, divided by the number of additional conflicting antecedents added to the cover. The set of examples that have the selected partial proof as one of their partial proofs are removed from the examples remaining to be covered. The process terminates when there are no examples remaining to be covered.

The EITHER implementation of the branch and bound algorithm is based on the recursive update of a *ready list*. The ready list is a list of elements each of which consists of two items: a partial cover, and the remaining examples not covered by the partial cover. Each remaining example is associated with its partial proofs according to the input theory. Initially, this list consists of a single element, with an empty partial cover and all of the failing positive examples as remaining elements. At each iteration the algorithm takes the first element in the ready list and expands it. The expansion is done by taking the first example in the element's remaining examples and adding a new element to the ready list whenever a partial proof for the example contains conflicting antecedents not already in the partial cover. The new element consists of the original partial cover augmented by the additional terms from the partial proof, and the remaining examples minus the first example. At the end of the expansion, the updated ready list is sorted. The sort uses a *lexicographic evaluation functional* (LEF) [Michalski, 1983] whose first criterion is the length of the partial cover and whose second is the number of remaining examples. In each case the LEF uses a less-than relationship. The recursion terminates when the element at the front of the ready list contains no remaining examples. That element is returned as the answer.

The details of the greedy algorithm and the branch-and-bound algorithm are presented in appendix B. In practice, EITHER uses the greedy cover algorithm for cover processing. EITHER also limits the number of partial proofs for each example, in practice, to three.

2.3.2 The Minimum Rule Cover

This section presents an overview of the minimum rule cover algorithm as well as a discussion of the two versions of the algorithm that have been implemented for EITHER.

Algorithm Overview. Finding the minimum rule cover is the first step in the theory specialization process for correcting overly general theories. An overly general theory manifests itself by having negative examples that are provable. The objective is to modify the theory in such a way that the negative examples are not provable, without losing any of the positive

examples. In analogy with the previous section, we would like to make the statement $\neg P$ true where

$$P \equiv P_1 \vee P_2 \vee \dots \vee P_n,$$

where

$$P_1 \dots P_n$$

are the proofs of all of the negative examples. We wish to express the proofs in terms of rules at the leaves of the proof trees (that is, rules whose antecedents include observables, see section 4.2 for justification).

For example, suppose our theory consists of rules for the concept "x":

$$r_1 : x \leftarrow y1 \wedge z$$

$$r_2 : x \leftarrow y2 \wedge z$$

$$r_3 : y1 \leftarrow b$$

$$r_4 : y2 \leftarrow c$$

$$r_5 : z \leftarrow d$$

and the negative examples are:

$$e1 : b \wedge c \wedge d,$$

$$e2 : b \wedge d,$$

then the rules at the leaves of the proof trees would be:

$$e1 : r_3 \wedge r_5, e1 : r_4 \wedge r_5, e2 : r_3 \wedge r_5.$$

In other words,

$$P \equiv (r_3 \wedge r_5) \vee (r_4 \wedge r_5) \vee (r_3 \wedge r_5).$$

We are looking for the minimum set of leaf-level rule retractions that will not allow any of the negative examples to be provable. Potentially the set of rule retractions could include all combinations of taking one rule from each disjunct (that is, the cartesian product of all of the terms contained in the disjuncts). Doing this for the example would result in the following covers:

$$r_3 \wedge r_4 \wedge r_3, r_3 \wedge r_4 \wedge r_5, r_3 \wedge r_5 \wedge r_3, r_3 \wedge r_5 \wedge r_5, r_5 \wedge r_4 \wedge r_3, r_5 \wedge r_4 \wedge r_5, r_5 \wedge r_5 \wedge r_3, r_5 \wedge r_5 \wedge r_5.$$

As can be seen, many of the rules contained in these covers are redundant, so that the length of the covers could be shortened. In addition, the number of logically unique covers is also much shorter than the total of potential covers. In particular, the unique covers which cover all of the examples are:

$$r_3 \wedge r_4, r_3 \wedge r_4 \wedge r_5, r_3 \wedge r_5, r_4 \wedge r_5, r_5.$$

The *minimal* set of covers for this set of examples would be $r_3 \wedge r_4, r_5$, where no cover in the minimal set is logically implied by any other cover. Finally, the minimum cover for this set of examples is r_5 , where a minimum cover is the shortest length rule-set that covers all of the examples.

Therefore, the first step in the minimum rule cover algorithm is to reduce the logical expression for the negation of proofs of the examples to a more tractable form. Let $\phi(P) = \neg P$. Let *factor* be the rule that is contained in the most proofs in P and that appears in at least two proofs. Let *factored* be those proofs in P that contain *factor*, with *factor* removed. Let *remainder* be the remaining proofs, those that do not contain *factor*. Then

$$P \equiv [factor \wedge factored] \vee remainder,$$

and

$$\phi(P) = [\neg factor \vee \neg factored] \wedge \neg remainder, \quad (2.1)$$

by De Morgan's law. Consequently,

$$\phi(P) = (\neg factor \wedge \neg remainder) \vee (\neg factored \wedge \neg remainder).$$

But note that

$$\phi(a \vee c) = \neg(a \vee c) = \neg a \wedge \neg c,$$

where a and c are arbitrary logical expressions. Hence,

$$\phi(P) = \phi(factor \vee remainder) \vee \phi(factored \vee remainder), \quad (2.2)$$

which is the recursive definition for the function used to eliminate common factors from P . The boundary conditions are: when $(factor = false) \wedge (factored = false)$, the rule retractions are the cartesian product of the negations of the rules found in the disjuncts of $remainder$; when one of the proofs in P is equal to $factor$, then $\phi(P) = \phi(factor \vee remainder)$.

The justification for the first boundary condition is that if all that is left is a set of proofs with no common factor, then this set is equivalent to:

$$(r_{11} \wedge r_{12} \dots r_{1n}) \vee \dots \vee (r_{n1} \wedge r_{n2} \dots r_{nm}),$$

where r_{ij} is the j th rule used in the i th proof in the proof set. The inverse of this expression is:

$$(\neg r_{11} \vee \neg r_{12} \dots \neg r_{1n}) \wedge \dots \wedge (\neg r_{n1} \vee \neg r_{n2} \dots \neg r_{nm}).$$

As a result, the second expression is satisfied by any model that takes one element from the first conjunct, one element from the second, and so on. Hence the set of possible models is the cartesian product of the set each of whose elements is the set of rule negations appearing in one of the conjuncts.

The justification for the second boundary condition is that if $factor$ appears in one of the terms of $factored$ (before it is factored), then the factored form will contain a term that is true. Therefore $factored$ will be true;

$$\phi(factored \vee remainder) = \phi(true) = false;$$

and

$$\phi(P) = \phi(factor \vee remainder)$$

from equation 2.2, above.

The function of the minimum rule cover algorithm is then to choose a minimum cover from among those corresponding to equation 2.2. Two versions of the minimum rule cover algorithm have been implemented for EITHER. One uses exhaustive search; the other uses greedy search.

The exhaustive rule cover algorithm. Equation 2.2 corresponds to a unique set of covers, any one of which is guaranteed to cause all of the provable negative examples to be unprovable. In order to select the cover used by the theory specialization algorithm, the set of covers from equation 2.2 is processed by a LEF that uses the following ordered metrics:

1. Number of *overlapping* examples. An example overlaps if it is both a positive and negative example for a given rule, which can only happen in multiple category theories (see Chapter 4 for details). In the single category case there cannot be overlapping examples since, using negation as failure, an example is either classified as an example of the category or it is not.
2. Number of unprovable positive examples when the cover is removed.
3. Cover length.

In each case the minimum value is selected. Ultimately, a cover containing no overlapping examples is obtained using the approach described in Chapter 4. Using the rules that cause the least number of positive examples to fail is motivated by a consideration of the *domain of applicability* for the proposed rules. The domain of applicability for a given rule is the set of examples that require the rule in their correct proof. EITHER selects the set of rules that correspond to the smallest domain of applicability for the theory as a whole. In the limit, if the domain of applicability is empty, then EITHER will be able to retract the rules in the cover without causing any positive examples to be unprovable. At the opposite extreme, if the domain of applicability were not considered, EITHER might select rules for the cover that were required for all proofs of all of the positive examples, such as the *stable* rule in the cup theory. Choosing the minimal cover length is another example of the Occam's razor heuristic, resulting in EITHER's changing the least number of rules in the input theory.

The greedy rule cover algorithm. EITHER can also choose a greedy algorithm for cover selection, derived from consideration of equation 3.1. Recall that equation 3.1 states:

$$\phi(P) = [\neg factor \vee \neg factored] \wedge \neg remainder.$$

Here, the meaning of $\neg factor$ is that *factor* is chosen as part of the rule cover, hence it can be removed from the input theory, hence $\neg factor$ will be true for any proofs that used to depend on *factor*. As a result, the entire expression can then be reduced to:

$$\phi(P) = \neg factor \wedge \phi(remainder).$$

In other words, at each iteration the algorithm finds a rule factor as in the original algorithm, but simply discards the factored terms, since these are all covered by the rule factor (that is, none of the examples corresponding to these terms will be provable when the rule factor is removed). At each iteration, the greedy algorithm selects as a factor the rule that participates in the greatest number of remaining proofs. Each proof containing the rule factor is then removed from the remaining proofs. The process terminates when there are no remaining proofs.

The details of the exhaustive and greedy rule cover algorithms are given in Appendix C.

2.4 Theory Generalization

Figure 2.6 shows the EITHER response to specialization errors in the input theory. EITHER first forms the minimum cover, as discussed in the previous section, that consists of

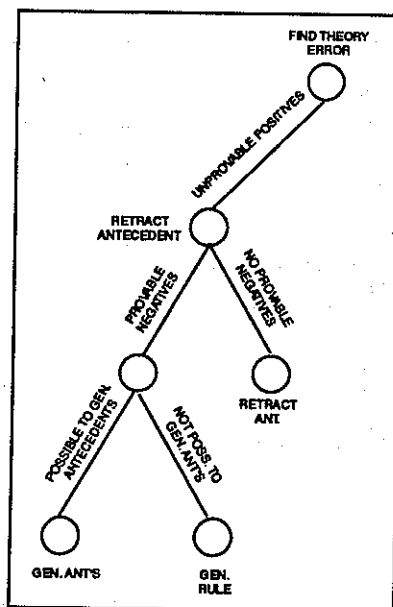


Figure 2.6: EITHER Generalization Response to Theory Errors

a set of conflicting antecedents such that, if all the antecedents in the set were retracted, all of the failing positive examples would be provable. The conflicting antecedents are associated with the rules in which they participate, with one or more conflicting antecedents per rule. Each such rule has associated with it the failing positive examples that used the rule in a partial proof. Each rule in the cover is sequentially generalized during theory generalization.

There are three possible generalization operators that can be used by EITHER in rule generalization. They are:

- antecedent retraction
- antecedent generalization
- inductive rule generalization.

To achieve conflict resolution, these operators are attempted in the order given. The motivation for this ordering, as well as the details of the operators, is given in the next three sections.

2.4.1 Antecedent Retraction

For each rule in the cover, the first step in the theory generalization process is to remove the conflicting antecedents associated with the rule. With one exception, if removing the conflicting antecedents does not result in any negative examples becoming provable when the modified rule is added to the theory then the generalization is complete, since removing a conflicting antecedent is the simplest possible syntactic change to the theory.

The exception occurs when the retraction causes all of the antecedents for the rule to be removed. In this case, rather than remove all of the antecedents, EITHER removes the consequent for the rule as an antecedent in all of the *parent rules* for the given rule. A

```

The selected antecedent cover is:
  ((cup-volume) <- (volume ?x) (<= ?x 12) (>= ?x 8))
  Conflicting antecedent:
    ((volume ?x) (<= ?x 12) (>= ?x 8))
  Antecedent cover complete.
Starting initial correction of leaf-level rules.
Tentatively removing conflicting antecedents
  ((volume ?x) (<= ?x 12) (>= ?x 8))
from rule ((cup-volume) <- (volume ?x) (<= ?x 12) (>= ?x 8))
new rule is: ((cup-volume) <-)
Change in theory due to popup:
Conflicting antecedent: ((cup-volume))
Rules:
  ((liftable) <- (graspable) (lightweight))
  Conflicting antecedent:
    ((cup-volume))
    ...
The rule
  (<- (liftable) (graspable) (lightweight) (cup-volume))
had the antecedent
  (cup-volume)
removed during popup, to become:
  (<- (liftable) (graspable) (lightweight))

```

Figure 2.7: Antecedent Popup Example

parent rule is a rule that appears at the level above the given rule in a proof tree: that is that uses the given rule in a partial proof. This process is called *antecedent popup*. The benefit of antecedent popup is that the correction is limited to just those rules that were associated with the failing positive examples. If the original rule were generalized by removing all of its antecedents, then any rule that used the consequent of the original rule as an antecedent would, in effect, be generalized. This generalization would be inadvertent in the case where the rule using the original rule was not present in any of the partial proofs of the failing positive examples.

As an example of antecedent popup, assume that the liftable rule is modified to add an antecedent that refers to an additional, unneeded rule (a leading "?" indicates a variable):

```

(liftable) ← (graspable) ∧ (lightweight) ∧ (cup-volume)
(cup-volume) ← (volume ?x) ∧ (≥ ?x 8) ∧ (≤ ?x 12)

```

Figure 2.7 shows the EITHER trace for this example.

2.4.2 Antecedent Generalization

However, if removing the conflicting antecedents results in provable negative examples, then doing so is an over-generalization to the theory. If so, the next step is to attempt to generalize the conflicting antecedents. Antecedent generalization is attempted before inductive rule generalization because it results in a smaller syntactical change to the rule. This is because antecedent generalization uses the same antecedent predicates as were used in the original rule, whereas inductive rule generalization will in general use entirely

(cup)	←	(stable) ∧ (liftable) ∧ (open-vessel)
(stable)	←	(has-bottom) ∧ (flat-bottom)
(liftable)	←	(graspable) ∧ (lightweight)
(graspable)	←	(has-handle)
(graspable)	←	(width small) ∧ (insulating)
(insulating)	←	(material styrofoam)
(insulating)	←	(material ceramic)
(open-vessel)	←	(has-concavity) ∧ (upward-pointing-concavity)

Figure 2.8: The Cup Theory With Two Missing Rules.

new predicates.

EITHER distinguishes between three types of antecedents for rules: linear, discrete, and boolean. A *linear antecedent* is one which specifies either a point or an interval for a numerical feature. A *discrete antecedent* is one which corresponds to a feature that has an enumerable set of values. A *boolean antecedent* corresponds to a propositional constant that can have the values true or false.

For linear antecedents, the generalization consists of extending the linear intervals that are present in the rule to be large enough to include the failing positive examples for the rule. For discrete antecedents, rules are added that contain the values for the feature that are present in the failing positive examples. For discrete antecedents, if all values for the feature are required to account for the failing positive examples, the feature is simply removed from the rule. For boolean antecedents, the generalization consists of removing the boolean antecedent from the rule.

Antecedent generalization is successful if no negative examples are provable with the generalized rules. Consequently, antecedent generalization is a *one-sided* generalization [Haussler, 1988]: only the failing positive examples are considered for the generalization, the negative examples are used simply to determine if the generalization was successful. Antecedent generalization results in a minimal change to the coverage provided by the theory, since any smaller change (smaller interval in the case of linear antecedents or less feature values in the case of discrete antecedents) would not succeed in covering all of the failing positive examples. When antecedent generalization introduces provable negative examples, rule formation is required. Rule formation is discussed later in this section.

As an example of a theory requiring antecedent generalization, consider the modification to the cup theory shown in Figure 2.8. The antecedent requiring generalization is the antecedent *material*, shown in the second italicized rule. The italicized rules should be interpreted as missing from the theory. The first highlighted rule will be used later to illustrate EITHER's response when a rule is missing from the theory. The trace in Figure 2.9 shows EITHER correcting the theory by generalizing the antecedent.

As an example of antecedent generalization of a discrete antecedent that actually results in the antecedent's being removed from the theory, consider a cup theory that has had an additional antecedent erroneously added to the liftable rule, say,

(liftable) ← (graspable) ∧ (lightweight) ∧ (color blue)

Figure 2.10 shows the EITHER trace for this case.

```

Processing antecedent (material ceramic)
in rule
((insulating) <- (material ceramic))
Attempting to generalize antecedent (material ceramic).
Old rule
(<- (insulating) (material ceramic))
has had antecedent (material ceramic) generalized to become:
(<- (insulating) (material ceramic))
(<- (insulating) (material styrofoam)).

```

Figure 2.9: Generalizing an Existing Antecedent

```

The selected antecedent cover is:
((liftable) <- (graspable) (lightweight) (color blue))
Conflicting antecedent:
((color blue))
Antecedent cover complete.
Starting initial correction of leaf-level rules.
Tentatively removing conflicting antecedent ((color blue))
from rule ((liftable) <- (graspable) (lightweight) (color blue))
new rule is: ((liftable) <- (graspable) (lightweight))

```

Figure 2.10: The EITHER Response to an Additional Antecedent

2.4.3 Inductive Rule Generalization

If negative examples become provable⁴, a more complex generalization of the rule is required. In this case, inductive rule generalization is performed by identifying a set of positive and negative examples for the rule and passing these to an inductive learner in order to form a set of additional rules with the same consequent as the original rule. In the event that one of the new rules is strictly more general⁵ than the original rule, the original rule is removed from the theory.

The set of positive and negative examples for the inductive rule formation is constructed through two processes: First, the positive examples are obtained as those that used the rule in a partial proof. Second, the failing negative examples are obtained by removing all of the antecedents from the rule and collecting the negative examples that become provable with the resulting theory. This second process is necessary because if our only goal was to cause the positive examples to become provable, a rule with no antecedents would suffice. Therefore, the antecedents that are added to the rule come about as a result of ensuring that no negative example that might rely on the consequent of the rule as an antecedent in a proof is provable with the updated rule. This is an example of proof by contradiction: we assume that the consequent of the rule is true and obtain the contradiction that a negative example is provable, implying that the negative example is not a member of the subconcept.

To illustrate this point, assume that the cup theory is missing the first rule for graspable:

(graspable) \leftarrow (has-handle).

The trace given in Figure 2.11 shows the EITHER response to this error. In this case examples 2 and 3 from Figure 2.3 are the positive examples for the induction, and examples 4, 5, and 6 are the negative examples.

2.5 Theory Specialization

Figure 2.12 shows the EITHER theory specialization response. When negative examples are provable, the theory needs to be specialized. The first step in correcting the theory is to form a cover consisting of those rules at the leaves of the theory that participated in the proofs of the negative examples. A minimum cover is formed (see Section 2.3.2). It consists of the minimum set of rules that, if retracted, would cause all proofs of the negative examples to fail. Each of the rules in the cover is processed sequentially for theory specialization.

EITHER uses the following specialization operators for rule specialization.

- rule retraction
- one-sided specialization

⁴Actually, receive new proofs in the case of multi-category theories, since, for such theories, an example may be a failing positive example for one category and a failing negative example for another. For single category theories, all failing negative examples are removed from the input to the generalization algorithm so that any proof of a negative example must be a new proof.

⁵One rule is strictly more general than another rule if both rules have the same consequent and the antecedents for the second rule logically imply the antecedents for the first rule, but the reverse is not true. For propositional theories this occurs when the set of antecedents for the first rule is a proper subset of those for the second rule.

Starting initial correction of leaf-level rules.
 Processing antecedent (width small)
 in rule
 ((graspable) <- (width small) (insulating))
 Attempting to generalize antecedent (width small).
 Generalizing antecedent resulted in negative examples being
 provable.
 Unable to generalize antecedent.
 Will have to form new rules.
 Learning new rules.
 Rules added to theory during generalization:
 (((graspable) <- (has-handle)))

Figure 2.11: Missing Rule Example

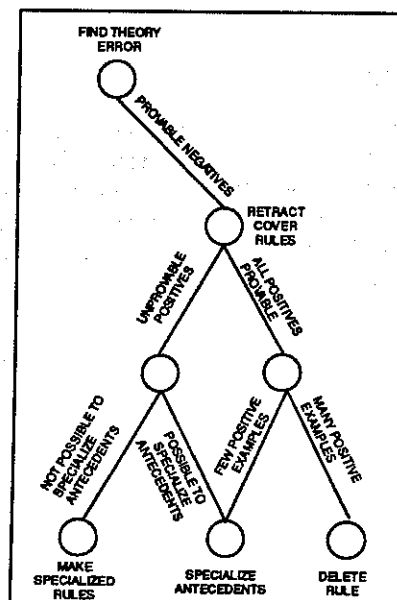


Figure 2.12: EITHER Specialization Response to Theory Errors

- inductive rule specialization

As with rule generalization, there is a total ordering on the operators sufficient to achieve conflict resolution. The details of the specialization operators and their ordering is described in the next three sections.

2.5.1 Rule Retraction

The first step in the specialization process is to remove the rule from the theory. If doing so results in failing positive examples, EITHER will either specialize the antecedents for the rule if possible (see next section) or form an inductive specialization for the rule (see Section 2.5.3). If no failing positive examples result from retracting the rule, EITHER checks to see if a sufficient number of positive examples have been seen. This condition is imposed so as not to rashly modify the theory by removing rules when only a small number of examples have been seen. In particular, with few examples, removing the rule may not cause any positive examples to fail simply because few or no positive examples have been seen at the time of the correction. Therefore, the exact form of the condition is that n positive examples have been seen, with the condition that $n > \frac{l}{\alpha}$, where l is the number of symbols in the rule to be retracted, α is a constant representing the best exclusion interval⁶ for the rule.

This condition requires that the number of positive examples be proportional to the length of the rule (that is, to the size of the syntactic change to the theory, since we will be retracting the rule), and inversely proportional to the efficiency of the possible antecedent specialization. Therefore, if the antecedent specialization would be particularly effective, requiring a small exclusion interval, rule retraction is not attempted unless many positive examples have been seen. Similarly, if the syntactic change to the theory is large, many positive examples are required before the change is made. If insufficient positive examples have been seen according to this criteria, EITHER will specialize the antecedents for the rule instead, which will always be successful since no such specialization can result in failing positive examples.

Figure 2.13 shows the EITHER trace when retracting an additional rule. In this case, the rule
 (stable) \leftarrow (has-handle)
 has been added to the theory.

2.5.2 One-Sided Specialization

The one-sided specializations attempt to specialize the rule *away* from the provable negative examples without considering the positive examples. Initially EITHER attempts to specialize the existing antecedents for the rule but, if this fails, EITHER specializes the rule by adding antecedents.

Specializing antecedents. EITHER initially tries to specialize one or more of the antecedents in the rule so that the negative examples associated with the rule will no longer

⁶For a discrete antecedent, an exclusion interval is the ratio of the number of values of the associated feature in the failing negative examples for the rule, to the total possible number of values for the associated feature. For linear antecedents representing an interval it is the ratio of the minimum subinterval necessary to exclude negative examples, to the total interval. It is not possible to specialize boolean features.

```

Starting correctability check on initial cover.
Rule
  ((stable) <- (has-handle)) was correctable
so am making correction to the theory.
Removing rule
  ((stable) <- (has-handle))
did not cause any positive examples to fail, so will do
minimal specialization.
The value for the selection metric, 7000000.0,
is greater than the length of the rule, 3,
so will retract the rule.

```

The value for the metric (α in the previous discussion) is initially set to a large value and is only changed if an antecedent specialization could be successful.

Figure 2.13: EITHER Response to Additional Rule

```

Rule
  ((cup-volume) <- (volume ?x) (>= ?x 8) (<= ?x 32))
was correctable so am making correction to the theory.
Removing rule
  ((cup-volume) <- (volume ?x) (>= ?x 8) (<= ?x 32))
caused positive examples to fail, so will form new rules.
Attempting minimal specialization.
Attempting to correct current antecedents.
Replacing with:
  (<- (cup-volume) (volume ?x) (>= ?x 8) (< ?x 16.1))
Minimal specialization was successful.

```

Figure 2.14: Specializing a Linear Antecedent

be provable. The rationale for attempting to specialize antecedents first is that under-specializing antecedents seems like the most likely error associated with writing rules, when compared to leaving out required antecedents, or adding an additional, unnecessary rule to the theory, or simply mis-writing the entire rule.

For linear antecedents, the specialization is to restrict the interval associated with the antecedent so as not to include any of the failing negatives. For example, assume that the cup theory has been modified to replace the *lightweight* antecedent in the *liftable* rule with *cup-volume*. Assume also that all cups are correctly between eight and sixteen ounces. The overly general rule for *cup-volume* which is added to the theory is that cup volume is between eight and thirty two ounces. The modifications are:

$$\begin{aligned}
 (\text{liftable}) &\leftarrow (\text{graspable}) \wedge (\text{cup-volume}) \\
 (\text{cup-volume}) &\leftarrow (\text{volume } ?x) \wedge (\geq ?x 8) \wedge (\leq ?x 32)
 \end{aligned}$$

The trace shown in Figure 2.14 shows EITHER's response to this error.

For discrete antecedents, the specialization is to exclude feature values. However, it is not sufficient to simply exclude those values found in the negative examples associated with the rule. There may be other negative examples which are not negative examples for the rule (that is, their feature value is not the value specified in the original rule), with feature

values which are not in the excluded interval. These examples can become provable using the updated rule if just the exclusion interval for the rule's negative examples is considered. In this case, EITHER checks the other examples and updates the exclusion interval to ensure that none are provable.

For boolean antecedents one would initially think that the correct specialization would be to negate the antecedent. However, as above, doing so might cause other negative examples to become provable (those which did not contain the given feature). In this case, if no other specialization is possible, EITHER attempts to delete the rule from the theory.

Adding antecedents. If it is impossible to specialize the antecedents associated with the rule, EITHER attempts to add additional antecedents to the rule as a further specialization. These antecedents can use features not currently present in the antecedents for the rule. The features that are selected have the minimum exclusion intervals corresponding to the negative examples. The selected antecedents are then added to the original version of the rule as a specialization.

2.5.3 Inductive Rule Specialization

If all of the previous specialization attempts fail, EITHER uses the inductive component to specialize the input rule. Failure is determined when all previous specializations result in failing positive examples with the proposed theory. In inductive specialization, EITHER associates positive and negative examples with the overly general rule and uses them to form a specialization for the rule. EITHER guarantees that the updated rule is indeed a specialization by removing the features associated with the current antecedents for the rule from the feature set used by the inductive learner in forming the rule updates. The current antecedents are then added back in to the updated rules returned by the inductive learner.

The negative examples for the rule are those examples which used the rule in an erroneous proof. The positive examples are examples which fail to be provable when the rule is removed from the theory. This categorization of examples is essentially the dual of the categorization used for inductive rule generalization, described in Section 2.4.

As an illustration of inductive rule specialization, assume that the cup theory has been modified such that both of the antecedents to the *stable* rule have been removed. Figure 2.15 shows the trace for this illustration. In this case, twenty positive examples and twenty near-miss negative examples were used in order to learn the correction.

2.6 Experimental Results

The EITHER algorithm was tested on a theory used for recognizing biological concepts in DNA sequences. The original theory is described in [Towell *et al.*, 1990]. Section D.2 presents the theory in its entirety. Summary information characterizing the theory is given below.

Number of rules: 11
 Number of consequents: 5
 Number of symbols: 76
 Average number of disjuncts: 2.20
 Average number of antecedents: 5.91

The purpose of the theory is to recognize *promoters* in strings composed of nucleotides (one of A, G, T, or C). A promoter is a genetic region which initiates the first

```

Starting correctability check on initial cover.
Rule
  ((stable) <-) was correctable
so am making correction to the theory.
Removing rule
  ((stable) <-)
caused positive examples to fail, so will form new rules.
Attempting minimal specialization.
Attempting to specialize rule by adding antecedents to the rule.
Replacing with: ((<- (stable) (not (has-concavity))))
Minimal specialization was unsuccessful,
so inductively specializing rule.
Adding rule
  (<- (stable) (flat-bottom) (has-bottom))
to theory.

```

Figure 2.15: EITHER Inductive Rule Specialization

step in the expression of an adjacent gene (*transcription*), by RNA polymerase. The input features are 57 sequential DNA nucleotides. The examples used in the tests consisted of 53 positive and 53 negative examples assembled from the biological literature. The initial theory classified none of the positive examples and all of the negative examples correctly, thus indicating that the initial theory was entirely overly specific.

Figure 2.16 shows the performance results obtained when EITHER was used to refine this theory. In each test, performance was measured against twenty five test examples. The number of training examples was varied from one to eighty, with the training and test examples drawn from the entire example population, with no overlap. The results were averaged over 21 samples.

Figure 2.16 shows that even though the DNA theory provided no initial performance advantage for EITHER (for ID3, a random class was chosen and all of the examples were assumed to be in that class), the fact that the EITHER modifications were relative to an existing theory quickly resulted in a significant performance advantage. After only ten training examples, EITHER was already performing at a level approximately six percent above ID3. The reason for including ID3 in the performance graphs is that ID3 represents what EITHER's performance would have been if it were not given an initial theory. This is because ID3 is the inductive component used by EITHER⁷. Therefore including the ID3 curve, as will be done for performance graphs throughout the remainder of the dissertation, provides a clear illustration of the advantage provided by theory-based learning. In fact, if a different inductive system were substituted for ID3 the absolute performance of both learning systems might change, but the relative advantage of EITHER compared to the purely inductive system should remain approximately the same.

A one-tailed Student t-test on paired differences showed that the superior performance of EITHER is statistically significant to at least the 5% level for every point plotted

⁷With no theory, EITHER would fail to classify any of the promoter examples. It would then add the rule (*promoter*) ← to the (non-existent) theory to identify negative examples. This would include all of the negative examples since all of the negative examples would be mis-classified with this rule. It would then send over all of the positive and negative examples to the inductive component (ID3) to learn a new rule for (*promoter*), which would duplicate ID3's operation in this case.

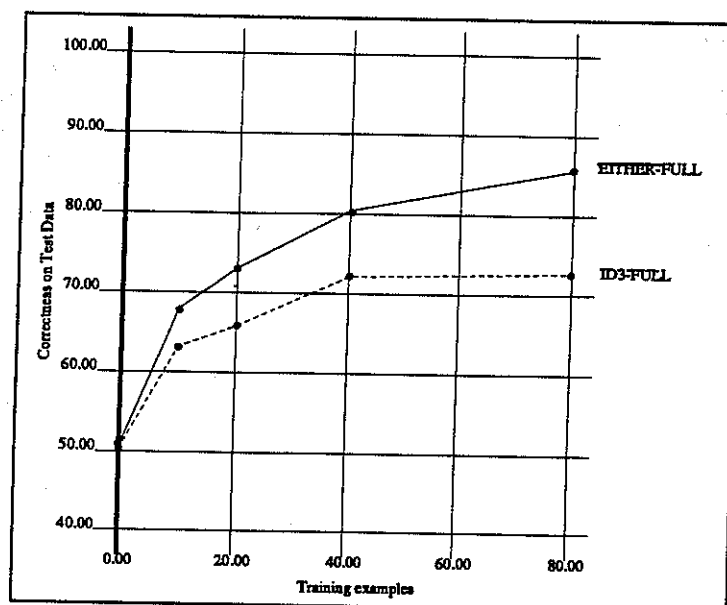


Figure 2.16: EITHER Results for the DNA Theory

on the learning curves. After 80 training examples, the 95% confidence interval for the difference between EITHER and ID3 is 10.201 to 15.806% (that is, with a probability of 0.95, EITHER's accuracy is between 10.201 and 15.806 percentage points higher than ID3's).

Another way of looking at the performance advantage provided by EITHER is to consider the additional examples required by ID3 in order to achieve equal performance with EITHER. For example, at seventy five percent performance, ID3 requires over sixty additional training examples to achieve equal performance with EITHER.

Appendix E shows a typical example of the kinds of modifications that EITHER made to the DNA theory. These revision include removing antecedents, generalizing antecedents, and popping up because all of the antecedents of a rule were removed. The concept *conformation* was removed from the *promoter* concept in its entirety. This correction was validated by the biologist associated with the KBANN project (Noordewier), who indicated that the conformation theory was a weakly-justified theory when it was originally introduced. The theory modifications generated by EITHER in this case also included rule compression due to both intra- and inter-compression, see Chapter 3.

The corrections to the rules that EITHER provided also seemed to cluster about the nucleotide positions associated with the original rules (that is, the tenth nucleotide position in the case of the minus.10 rules and the thirty fifth nucleotide position in the case of minus.35 rules). This seems to indicate that the original concept that promoter sequences are indicated by particular nucleotide configurations within certain *regions* of the nucleotide chain is valid, although the original rules themselves were incorrect.

This domain theory was also tested on the KBANN system ([Towell *et al.*, 1990]), which translates the initial theory into an equivalent neural net, and then learns by applying the Rumelhart backpropagation algorithm [Rumelhart *et al.*, 1986] to the resultant neural net. KBANN obtains a performance which is significantly better than EITHER (a test set accuracy of 92% with 105 training examples) in this domain. A possible explanation for the performance advantage is that the DNA task involves learning a concept of the form *N out of*

these M predicates must be true. [Fisher and McKusick, 1989] report that the backpropagation algorithm is particularly well suited to problems involving learning N out of M functions. Some aspects of the promoter concept fit the N out of M format where, for example, there are several potential sites where hydrogen bonds can form between the DNA and the protein; if enough of these bonds form, promoter activity can occur. On the other hand, for EITHER to learn this concept it would have to learn separate Horn clause rules for each potential configuration by deleting different combinations of antecedents from the current rule, which makes this a comparatively difficult learning task for the EITHER algorithm.

Chapter 3

Revising Intermediate Rules

EITHER's initial bias is to revise rules at the leaves of the theory tree (see Section 4.2 for justification). However, EITHER is also capable of identifying and correcting theory errors that occur at interior portions of the theory tree. Rules located interior to the theory tree are called *intermediate rules*. In the general case, EITHER uses a hill-climbing technique to identify the theory correction that represents the smallest syntactic change to the theory. Once again, this is based on the Occam's razor heuristic of preferring to make the least change necessary to render the theory consistent with the examples.

The process of learning intermediate rules is composed of two parts. The first part, *consequent identification*, consists of locating the best candidate for the left hand side, or consequent, of the learned rule. This process is described in Section 3.1. The second part, *antecedent construction*, identifies the set of antecedents to be used in the new rule. This set of antecedents may include the consequents of higher level rules, so that both the consequent and antecedents of the corrected rule may actually correspond to intermediate concepts in the original theory. The process of selecting and constructing the rules used by the antecedents of the new rule is described in Section 3.2.

3.1 Consequent Identification

There are three mechanisms involved in the process of consequent identification. The first two affect the starting point for consequent selection, and the third determines the actual consequent selected. The process of consequent identification starts at the leaves of the theory tree using the rules identified in the minimum cover. The first factor affecting the starting point for consequent identification is the need to identify a *correctable* cover. (The requirement for a correctable cover only applies to multiple category theories and is discussed in Section Chapter 4). Choosing a correctable cover may result in higher level nodes being selected for the rule cover. The second factor that can affect the location of the consequent pertains when there are *gaps* in the theory. In this case there is no rule available to prove an antecedent in a higher level rule (for example, if the rule for *liftable* was missing from the *cup* theory). When this happens a conflicting antecedent is identified for the higher level rule that requires the missing rule in its proof. The indication of this situation is that the minimum antecedent cover initially contains intermediate concepts.

Once the starting point has been chosen using the techniques described above, a third mechanism is employed to actually select the consequent to be used in the theory correction. The consequent is selected from among the consequents associated with the rules in the *proof chain*¹. A simple heuristic is used to identify the location for the correction.

¹A proof chain is a single path through a proof, starting with the rule whose consequent is the goal for the proof and ending with the rule selected as the starting point in the consequent identification process.

The consequent is selected that results in the syntactically simplest set of rules using the antecedent construction techniques described below.

Finding the location for the correction is done iteratively, using a hill climbing technique. At each iteration, the complexity of the required corrections is compared between the current rule and the rule's parents. All rules having a common parent are grouped together, and their complexity is considered as a whole in the comparison. The reason for this grouping is that the single correction to the parent rule accounts for all of the corrections to the current rules.

The process is terminated whenever one of two conditions applies: either the complexity of the parent corrections is equal to or greater than the complexity of the current correction, or the current rule set consists of a single rule with multiple parents. The second constraint is a simple heuristic motivated by the fact that the correction to a single rule having multiple parents is likely to be simpler than the correction to the parent rules.

In order to illustrate the intermediate rule learning process, we will use a theory for animals, given in Figure 3.1. This theory is also used in Chapter 5 to demonstrate the learning of shared rules. The animal theory is an extended version of a set of rules given in [Winston, 1989, pages 388-390]. For testing and training purposes thirty examples of each category were generated for use with the animal theory. This was done by first forming *core examples*, which contain just the observables needed to complete a proof. For linear features, a value was chosen randomly from the range required for a proof. A core example was formed for each possible proof for each category. For example, below are the core examples of ducks.

1. (body-covering feathers) (foot-type webbed) (fly)
2. (birth egg) (foot-type webbed) (fly)

Next, random values for the remaining observable features were added to the core examples to create full examples. However, adding random values can sometimes make an example provable in another category as well. Consequently, each example was checked to make sure it was provable in only one category before adding it to the final data set. A total of 360 examples of animals and 210 examples of computers were created in this manner.

Returning to the problem of consequent identification, assume that the first rule for (ungulate) is missing the antecedent (foot-type hoof). This is an overly general theory, since any mammal is considered to be an ungulate. The trace shown below was generated by EITHER using giraffes as examples of ungulates and cheetahs as examples of non-ungulates. The trace shows how EITHER first attempts to make corrections to the leaf-level rules but ultimately selects the ungulate rule for correction, since the correction is simpler.

Test results for thirty training examples:

Training EITHER...

Original theory and thirty training examples:

Theory classified 46.666668% of the training examples correctly.

There are provable negative examples so will specialize theory.

Finding minimum rule cover.

27 proofs remain to be covered.

15 proofs remain to be covered.

4 proofs remain to be covered.

0 proofs remain to be covered.

Rule cover complete.

EITHER initially tries to make corrections to the leaf-level rules for mammal.
--

(mammal)	←	(body-covering hair)
(mammal)	←	(feed-young milk)
(mammal)	←	(birth live)
(bird)	←	(body-covering feathers)
(bird)	←	(birth egg) (fly)
(ungulate)	←	(mammal) (foot-type hoof)
(ungulate)	←	(mammal) (ruminant)
(carnivore)	←	(eat-meat)
(carnivore)	←	(teeth pointed) (foot-type clawed)
(giraffe)	←	(ungulate) (neck-length ?n) ($\geq ?n 5$) ($\leq ?n 6$) (color tawny) (pattern spots) (pattern-color black)
(zebra)	←	(ungulate) (color white) (pattern stripes) (pattern-color black)
(cheetah)	←	(mammal) (carnivore) (color tawny) (pattern spots) (pattern-color black)
(tiger)	←	(mammal) (carnivore) (color tawny) (pattern stripes) (pattern-color black)
(dolphin)	←	(mammal) (fore-appendage fin) (color gray) (body-covering moist-skin) (body-length ?b) ($\geq ?b 4$) ($\leq ?b 6$)
(whale)	←	(mammal) (fore-appendage fin) (color gray) (body-covering moist-skin) (body-length ?b) ($\geq ?b 10$) ($\leq ?b 60$)
(bat)	←	(mammal) (color black) (pattern none) (pattern-color none) (fly)
(platypus)	←	(mammal) (birth egg) (foot-type webbed)
(ostrich)	←	(bird) (not (fly)) (neck-length ?n) ($\geq ?n 3$) ($\leq ?n 4$) (color white) (pattern patch) (pattern-color black)
(penguin)	←	(bird) (color white) (pattern patch) (pattern-color black) (foot-type webbed) (not (fly))
(duck)	←	(bird) (foot-type webbed) (fly)
(grackle)	←	(bird) (color black) (pattern none) (pattern-color none) (fly)

Observable Features: feed-young body-covering birth eat-meat fly teeth fore-appendage foot-type neck-length body-length color pattern pattern-color ruminant

Categories: giraffe zebra cheetah tiger dolphin whale bat platypus penguin ostrich duck grackle

Figure 3.1: The Complete Animal Theory

Starting correctability check on initial cover.

Rule
 ((mammal) <- (feed-young milk)) was correctable
 so am making correction to the theory.

Removing rule
 ((mammal) <- (feed-young milk))
 caused positive examples to fail, so will form new rules.
 Adding rules ((mammal) <- (feed-young milk) (foot-type hoof)))
 to theory.

Rule
 ((mammal) <- (body-covering hair)) was correctable
 so am making correction to the theory.

Removing rule
 ((mammal) <- (body-covering hair)) caused positive examples
 to fail, so will form new rules.
 Adding rules ((mammal) <- (body-covering hair) (ruminant)))
 to theory.

Rule
 ((mammal) <- (birth live)) was correctable
 so am making correction to the theory.

Removing rule
 ((mammal) <- (birth live)) caused positive examples to fail,
 so will form new rules.

Adding rules
 ((mammal) <- (birth live) (neck-length ?g8193
 (>= ?g8193 4.637165)))
 to theory.

Correctability check complete.

EITHER next checks the corrected leaf level rules against their parent (the erroneous ungulate rule.)

Starting syntactical simplicity check.

Comparing corrections to
 ((mammal) <- (feed-young milk))
 ((mammal) <- (body-covering hair))
 ((mammal) <- (birth live))
 against the proposed changes to their parent
 ((ungulate) <- (mammal))
 for syntactical simplicity
 The correction to the parent is:
 ((ungulate) <- (foot-type hoof) (mammal))

Since the syntactical change to the parent is less than the changes to the original rules, **EITHER** makes the change to the parent rule.

Popping up due to syntactical simplicity, replacing

((mammal) <- (body-covering hair) (ruminant))
 ((mammal) <- (feed-young milk) (foot-type hoof))
 ((mammal) <- (birth live) (neck-length ?g8193
 (>= ?g8193 4.637165)))

with

((ungulate) <- (mammal) (foot-type hoof))
 in the cover

Syntactical simplicity check complete.

Final theory:

(mammal)	←	(body-covering hair)
(mammal)	←	(feed-young milk)
(mammal)	←	(birth live)
(bird)	←	(body-covering feathers)
(bird)	←	(birth egg) (fly)
(ungulate)	←	(mammal) (foot-type hoof)
(ungulate)	←	(mammal) (ruminant)
(carnivore)	←	(eat-meat)
(carnivore)	←	(teeth pointed) (foot-type clawed)
(predator)	←	(mammal)(carnivore)
(predator)	←	(hunts-for-sport)
(hunts-for-sport)	←	(stalks-prey)(human)

Figure 3.2: Modified Animal Theory Segment

```
((mammal) <- (body-covering hair))
((mammal) <- (feed-young milk))
((mammal) <- (birth live))
((ungulate) <- (mammal) (ruminant))
((ungulate) <- (mammal) (foot-type hoof))
```

Popup due to syntactical simplification can also occur during theory generalization. To illustrate this possibility, consider the theory in Figure 3.2, that is a slightly modified segment of the animal theory. Rules for *predator* have been added to the original segment of the theory. We assume that the second rule for predator, shown in boldface, is missing from the incorrect version of the theory, causing it to be overly specialized. This rule states that a predator is a human being who hunts for sport. The trace below illustrates how EITHER detects and corrects this problem. EITHER also connects the new rule to hunts-for-sport, using a form of antecedent construction, discussed in the next section. The examples for predator were generated as described above for the animal categories, and examples of ungulate were used as counter-examples. Thirty examples of predator and thirty examples of ungulate were used to generate the trace.

Starting initial correction of leaf-level rules.

EITHER starts the initial check against the failing leaf-level rules for mammal and carnivore.

```
Tentatively removing conflicting antecedent ((eat-meat))
from rule ((carnivore) <- (eat-meat))
new rule is: ((carnivore) <-)
Negative examples were provable
with the conflicting antecedents removed.
Attempting to generalize conflicting antecedent.
Unable to generalize conflicting antecedent
so will form new rules instead.
Learning new rules.
Rules added to theory during generalization:
(((carnivore) <- (hunts-for-sport)))
Tentatively removing conflicting antecedent ((birth live))
```

from rule ((mammal) <- (birth live))
 new rule is: ((mammal) <-)
 Negative examples were provable
 with the conflicting antecedents removed.
 Attempting to generalize conflicting antecedent.
 Unable to generalize conflicting antecedent
 so will form new rules instead.
 Learning new rules.
 Rules added to theory during generalization:
 (((mammal) <- (hunts-for-sport)))

It next checks against the parent of these rules, which is the predator rule.

Starting syntactical simplicity check on initial rule set.
 Comparing corrections to
 ((carnivore) <- (eat-meat))
 ((mammal) <- (birth live))
 against the proposed changes to their parent
 ((predator) <- (mammal) (carnivore))
 for syntactical simplicity
 The correction to the parent is:
 ((predator) <- (mammal) (carnivore))
 ((predator) <- (hunts-for-sport))
 Popping up due to syntactical simplicity, replacing
 ((carnivore) <- (eat-meat))
 ((carnivore) <- (hunts-for-sport))
 ((mammal) <- (birth live))
 ((mammal) <- (hunts-for-sport)) with
 ((predator) <- (mammal) (carnivore))
 ((predator) <- (hunts-for-sport))
 in the cover
 Beginning rule reduction process.
 Rule reduction complete.
 Added rules:
 ((predator) <- (hunts-for-sport))
 Summary of EITHER theory changes:
 The rule
 ((predator) <- (mammal) (carnivore))
 was generalized as follows:
 by adding the following rule: ((predator) <- (hunts-for-sport))
 Final theory:
 ((mammal) <- (body-covering hair))
 ((mammal) <- (feed-young milk))
 ((mammal) <- (birth live))
 ((bird) <- (body-covering feathers))
 ((bird) <- (birth egg) (fly))
 ((ungulate) <- (mammal) (foot-type hoof))
 ((ungulate) <- (mammal) (ruminant))
 ((carnivore) <- (eat-meat))
 ((carnivore) <- (teeth pointed) (foot-type clawed))
 ((predator) <- (mammal) (carnivore))
 ((predator) <- (hunts-for-sport))
 ((hunts-for-sport) <- (stalks-prey) (human))

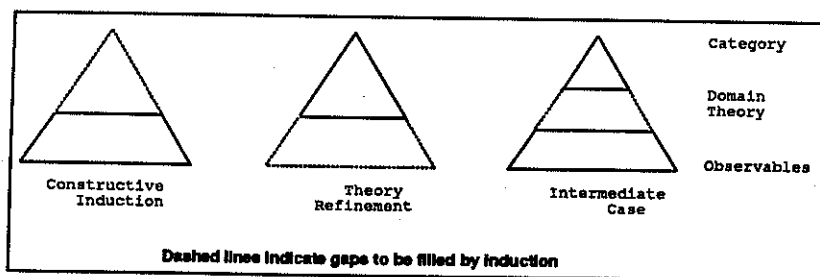


Figure 3.3: Different Types of Gaps in Incomplete Theories

(cup)	←	(stable) ∧ (liftable) ∧ (open-vessel)
(stable)	←	(has-bottom) ∧ (flat-bottom)
(liftable)	←	(graspable) ∧ (lightweight)
(graspable)	←	(has-handle)
(graspable)	←	(width small) ∧ (insulating)
(insulating)	←	(material styrofoam)
(insulating)	←	(material ceramic)
(open-vessel)	←	(has-concavity) ∧ (upward-pointing-concavity)

Figure 3.4: The Cup Theory

3.2 Antecedent Construction

Research in constructive induction frequently assumes that the existing domain theory consists of a number of rules that define a set of intermediate concepts (derived features) in terms of observables. The rules connecting these intermediate concepts to the categories are assumed to be missing and must be learned using induction. The theory is used to derive truth values for all of the intermediate concepts and these are used as additional input features for induction of the category rules. This is the first situation illustrated in Figure 3.3 where there is a gap at the “top” of the theory. For example, imagine the category rule for concluding *cup* was missing from the cup theory shown in Figure 3.4 (originally introduced in Chapter 2 and repeated here for convenience).

Some research in refinement of incomplete theories with missing rules [Danyluk, 1989a] assumes that the domain theory has correct rules for inferring categories from intermediate concepts but is instead missing rules connecting observables to intermediate concepts. Partial explanations (incomplete proofs) are used to isolate intermediate concepts that should be provable for some examples but are not. Induction is then used to learn rules for inferring these intermediate concepts from observables. This is the second situation illustrated in Figure 3.3 where there is a gap at the “bottom” of the theory. For example, imagine one of the rules for inferring *graspable* was missing from the cup theory.

A third case is illustrated in the final situation in Figure 3.3 where there is a gap in the “middle” of the theory. For example, imagine the rule for inferring *liftable* was missing from the cup theory. None of the previous research seems to directly address this issue.

An ideal system should be able to deal with multiple gaps occurring at arbitrary levels in the domain theory. It should also be able to introduce new intermediate concepts

in order to handle the situation in which the gap in the theory spans multiple levels. For example, imagine that all rules for inferring both *liftable* and *graspable* were missing from the theory in Figure 3.4. In this case, the intermediate concept *graspable* is not even present in the theory and must be created.

EITHER combines a number of previous techniques from theory refinement and constructive induction in order to deal with this general problem. In particular, *concept utilization* employs existing rules in the theory to derive high-level features from the data. *Rule reduction* employs inverse resolution to introduce new intermediate concepts in order to fill a gap in the theory than spans multiple levels.

In concept utilization EITHER identifies intermediate concepts in the theory to be used as antecedents in the rule that is being learned. This identification is done by selecting those consequents that most strongly correlate with the class of the examples used to create the rule, where the example class is either positive or negative. The inductive rule reduction algorithm looks for patterns among the rules returned by the inductive learner, and promotes to higher level rules those patterns that occur most frequently.

3.2.1 Concept Utilization

The concept utilization algorithm uses forward chaining on the failing examples to identify intermediate level concepts that are deducible from the facts of the examples. The consequents of these rules are then fed to the inductive learner as additional features pertaining to the examples. In this way, if a higher level consequent is highly correlated with the class of the failing examples, the consequent is returned as an antecedent in the rules formed by the inductive learner.

For example, assume that the cup theory is missing the rule for *liftable*. Then forward chaining on the unprovable positive examples (in this case, all of the positive examples) will always add the feature *graspable* to the examples, since *graspable* is true for all positive examples. On the other hand, no negative example will deduce both *graspable* and *lightweight*, since no negative example is *liftable*. The negative examples become provable during the rule formation phase when *liftable* is assumed true. Consequently the inductive learner, given enough examples, will select the higher level consequent *graspable* as an antecedent for the new rule for *liftable*. The observable *lightweight* is also chosen because of the same effect. The trace shown in Figure 3.5 was generated using the cup examples defined in Chapter 2.

Concept utilization also allows EITHER to handle gaps at the very top of the theory as in normal constructive induction. For example, below is a trace of the system learning the missing rule for *cup* given 30 random positive and negative examples of cups.

Starting initial correction of leaf-level rules.

```
Processing antecedents in rule
((CUP) <-)
Unable to generalize antecedents.
Negative examples were provable with the antecedents removed.
Will have to form new rules.
Learning new rules.
Rules added to theory during generalization:
((CUP) <- (OPEN-VESSEL) (LIFTABLE) (STABLE)))
```

```

Starting initial correction of leaf-level rules.
Processing antecedent (liftable)
in rule
((cup) <- (stable) (liftable) (open-vessel))
Generalizing antecedent resulted in negative examples
being provable.
Unable to generalize antecedent.
Negative examples were provable with the antecedent removed.
Will have to form new rules.
Learning new rules.
Rules added to theory during generalization:
(((liftable) <- (lightweight) (graspable)))

```

Figure 3.5: Using Intermediate Concepts

3.2.2 Rule Reduction

The goal of rule reduction is to simplify the inductively-generated rules by making explicit the structure inherent in the revised rules. This process serves the twin purposes of compressing the rulebase and identifying new intermediate concepts. The rule reduction algorithm used by EITHER is based on the inverse resolution technique of Muggleton and Buntine [Muggleton and Buntine, 1988]. In particular, EITHER uses the intra-construction, inter-construction, and absorption operators for identifying more general rules.

The reduction operators. In DUCE [Muggleton, 1987], sets of rules are compared in order to identify common patterns, and then combined and compressed using one of the inverse resolution operators. The inter-construction and intra-construction operators introduce new concepts in the process. The basic procedure is an iterative one in which operators are applied repeatedly until no further reduction of the theory is possible.

In the inter-construction technique, a single rule is formed to extract the common pattern associated with the input rules. For example, rules such as $x \leftarrow w \wedge y \wedge z$ and $x \leftarrow u \wedge y \wedge z$ are combined to form the rules: $x \leftarrow w \wedge v$ and $x \leftarrow u \wedge v$ and $v \leftarrow y \wedge z$, where v is a new intermediate concept.

In intra-construction, new rules are formed representing the differences between the input rules. For example, the same rules as above would be combined as $x \leftarrow v \wedge y \wedge z$ where $v \leftarrow w$ and $v \leftarrow u$ where v is again a new intermediate concept. Note that unlike inter-construction, intra-construction requires that both input rules have the same consequent. The choice of whether to use inter-construction or intra-construction is dependent on the syntactic simplicity of the resultant update.

Absorption occurs when all of the antecedents for one rule (e.g. $x \leftarrow a \wedge b$) are contained in the antecedents of another (e.g. $y \leftarrow a \wedge b \wedge c$). The consequent for the smaller rule is inserted into the antecedents for the larger rule, in place of the antecedents that the two rules have in common (e.g. $y \leftarrow x \wedge c$). In the general case, absorption could happen even if there were many rules implying the consequent for the smaller rule, and the combination would represent a generalization to the larger rule. Since the original EITHER algorithm guarantees consistency with the example set, EITHER only allows absorption when there is a single version of the absorbed rule (i.e. a single rule with the given consequent) so that the semantics of the rules are unchanged.

EITHER's use of Inverse Resolution. After EITHER produces a revised theory that is consistent with the training examples, the above operators are used to compress any rules that were modified or created during the revision. In the process, new intermediate concepts are created. The EITHER procedure is slightly different from the original one in DUC in that it does not employ an oracle, does not actually generalize the input rules, and employs hill-climbing rather than best-first search in order to find a good operator to apply.

Let the original set of rules under consideration for rule reduction be given by

$$X_i \leftarrow A \wedge N_i \quad (1 \leq i \leq n)$$

where A represents the set of antecedents that are in common among all of the rules, and N_i represents the remaining antecedents for each rule. The objective in choosing A is to produce the greatest syntactic reduction. The intra-construction operator will compress the rules into the following set (recall that intra-construction requires that $X_i = X$ for all i):

$$\begin{aligned} X &\leftarrow A \wedge Y \\ Y &\leftarrow N_i \quad (1 \leq i \leq n). \end{aligned}$$

For the same set of rules, inter-construction produces

$$\begin{aligned} Y &\leftarrow A \\ X_i &\leftarrow Y \wedge N_i \quad (1 \leq i \leq n). \end{aligned}$$

We wish to choose the operator that provides the greatest syntactic reduction. The number of literals contained in the input rules is given by:

$$S = \sum_{i=1}^n (1 + m + \xi_i)$$

where ξ_i is the number of literals associated with N_i , and m is the number of literals associated with A . Hence,

$$S = n + nm + k, \text{ where } k = \sum_{i=1}^n \xi_i.$$

By a similar analysis, the number of literals in the intra-reduced rules is given by:

$$I_{intra} = n + m + k + 2.$$

The change due to intra-reduction is therefore $S - I_{intra} = nm - m - 2$. The reduction will be positive (i.e. the rules will have been simplified) whenever $n > \frac{m+2}{m}$. This implies the reduction will be non-negative whenever $n > 3$, for $m = 1$, and for any value of n when $m > 1$ (since we must always have two or more rules in order to attempt a reduction).

The number of literals in the inter-reduced rules is given by:

$$I_{inter} = m + 2n + k + 1.$$

The change due to inter-reduction is therefore $nm - n - m - 1$. This value will be positive whenever $n > \frac{m+1}{m-1}$. This implies the following constraints on inter-reduction

$$m = 1: \text{ no reduction possible, } m = 2: n > 3, m = 3: n > 2.$$

The constraints on the inter- and intra-construction operators are therefore:

$$\text{inter: } nm \geq 6, \text{ intra: } nm \geq 3.$$

We include the neutral points (reductions that produce the same number of literals as the original rules) since we are interested in discovering new intermediate concepts.

The computation of A is done separately for inter and intra reduction, since a different set of rules may be involved in each case. The computation is done using a greedy hill climbing algorithm. At each iteration, a new literal that causes the largest reduction in the input rules is chosen to add to A . If the reduction with the literal added is less than the previous reduction, the process halts. The overall process is halted when no further

Starting initial correction of leaf-level rules.

Processing antecedent (liftable) in rule

```
((cup) <- (stable) (liftable) (open-vessel))
```

Unable to generalize antecedent.

Negative examples were provable with the antecedent removed.

Will have to form new rules.

Learning new rules.

Rules added to theory during generalization:

```
((liftable) <- (has-handle) (weight ?g0009)
  (< ?g0009 1.1257166))
```

```
((liftable) <- (insulating) (width small) (not (has-handle))
  (weight ?g0009) (< ?g0009 1.1257166)))
```

The rules were reduced using inverse resolution to:

```
((liftable) <- (intra-0010) (weight ?g0009)
  (< ?g0009 1.1257166))
```

```
((intra-0010) <- (has-handle))
```

```
((intra-0010) <- (insulating) (width small) (not (has-handle)))
```

Figure 3.6: Reducing Rules

reduction is possible. Once A has been computed for each case, the reduction operator that produces the greatest syntactic reduction is chosen. In case of ties, intra-construction is chosen because it focuses the reduction on rules having a single consequent.

As an example of rule reduction, consider the case in which all of the rules for both *liftable* and *graspable* are deleted from *cup* theory. The trace shown in Figure 3.6 shows EITHER refining this theory using 50 examples. The intermediate concept INTRA-0010 formed using intra-construction is EITHER's new concept for *graspable*. The extra (not (has-handle)) antecedent on the second rule is a side-effect of translating ID3 decision trees into rules. It does not effect the semantics of the new concept and could be deleted using the sort of rule simplification methods discussed in [Quinlan, 1987].

Chapter 4

Multiple Category Theories and the Correctability Problem

The purpose of the EITHER system is to correct the initial theory such that the following statements are true for every example:

$$T \cup E \models C_E, \quad (4.1)$$

$$\forall C_i (C_i \neq C_E \Rightarrow T \cup E \not\models C_i) \quad (4.2)$$

where T represents the corrected theory, E represents the facts pertaining to any example in the training set, C_E is the category attached to the example (that is, C_E is the correct classification for the example), and C_i is any arbitrary category associated with the examples. For single category theories such as the cup theory of Chapter 2, the category for the examples is either the goal concept (category) for the theory, for example, *cup*, or *negative*, corresponding to $\neg \text{cup}$. Using negation-as-failure, it is assumed that the category $\neg C_E$ is a consequence of the theory when the category C_E cannot be derived. In the single category case, an example is inconsistent with the theory whenever the goal concept can be proven by the theory and the example is negative, or when the goal concept cannot be proven by the theory and the example is positive.

In the multiple category problem, both the theory and the examples represent multiple categories. In this case, C_E in equation 4.1 refers to the set of categories that can be deduced from the theory given the facts of the example. The EITHER algorithm is based on the assumption that each example is a member of a single category, when noise is not present in the examples (see Chapter 6 for a discussion of the noise-contaminated case). With this assumption, when noise is not present in the examples inconsistency can only occur when the set of categories generated by EITHER for the example does not consist of the single category actually associated with the example; that is, the category to which it belongs¹.

The presence of multiple categories introduces several complications into the theory revision problem. For example, the classification errors associated with examples can now have any of the following characteristics:

- An example is not classified in its own category. (These are what were called failing positive examples in the single category case.)
- An example is classified in a category to which it does not belong. (These are what were called failing negative examples in the single category case - that is, the example should be a negative example for the category but instead is provable in the category.)

¹We include as a possible category from EITHER the category *negative*, that occurs when no category is provable for the example.

- An example is not provable in its own category and is provable in one or more other categories. In this case the initial theory would be both overly specific and overly general.

This should be contrasted with the single category case, where the possible errors are that a positive example can fail to be proven or a negative example can be proven. In the single category case, an example is *always* assigned to a single category (that is, either positive or negative) by the theory because of the negation as failure assumption used by EITHER.

By analogy with the single category case, we will call an example a failing positive example if it is not provable in its own category. It will be called a failing negative example if it is provable in another category. Note however that it is now possible for the same example to be both a failing positive and a failing negative.

As will be shown below, the main complication that is introduced by multiple categories is that EITHER must now choose rules for update that are *correctable*. For a rule to be correctable the set of provable negatives and failing positive examples associated with a rule must not overlap when

- the consequent of a rule in the minimum cover is assumed true² in the case of theory generalization, or
- a rule in the cover is retracted, in the case of theory specialization.

EITHER requires the following pre-conditions on the training examples sent to the generalization and specialization algorithms, to help ensure correctability:

- there are no failing negative examples input to the generalization algorithm,
- there are no failing positive examples input to the specialization algorithm,
- each example corresponds to a single category.

The reasons for these constraints will be made clear in the remainder of the chapter.

4.1 The Reasons for the Correctability Problem

As mentioned in Chapter 2, the rule correction process in EITHER starts with a cover (either the minimum antecedent cover in the case of overly general theories, or the minimum rule cover in the case of overly specialized theories). Initially, the cover consists of leaf-level rules. In certain cases, it may not be possible to make a correction to a leaf-level rule that can eliminate the correctability problems described above.

For example, consider Figure 4.1, which shows a simple theory in which the category rules (rules that have as a consequent one of the defined categories) rely on the same leaf-level rule. This is an extremely pathological theory in that any example will either be provable in both categories or neither category, and the same remarks apply when any *change* is made to the leaf-level rule. As a result, the "R" rule cannot be changed to correct the theory problem. In general, the problem is detecting that such a condition exists (either

²That is, the rule is replaced by a rule with the same consequent and no antecedents.

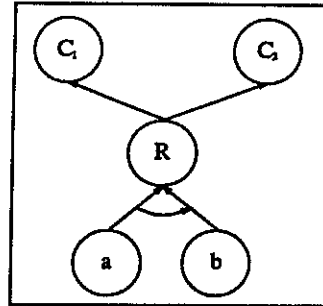


Figure 4.1: A Simple Inconsistent Theory

directly or indirectly), and then making a correction that causes examples to be classified only in their own categories.

To illustrate the impact on EITHER processing, suppose we have a C_1 example which is provable as C_1 using the initial theory. Therefore, the example also will be provable in category C_2 , meaning the theory is overly general. Removing the "R" rule (the first attempted step in the specialization process) will cause the example to fail in category C_2 , but will also cause it to fail in its own category. What this means is that the same example is both a positive example (requires the "R" rule in a proof of C_1), and negative example (the example is provable in C_2 using the "R" rule) for the specialization to the "R" rule.

If the "R" rule were overly specialized, then the example would fail in C_1 , but would become provable in C_2 when the antecedents of the rule were removed. Again, the same example would be both a positive and negative example for the required generalization to the rule.

Examples that show such behavior during the theory correction process are called *overlapping*. That is, overlapping examples are members of both the positive examples and negative examples for the given rule.

Associating these examples with a particular rule is the reason why the theory input to the specialization algorithm is required to be strictly overly general, and the input to the generalization algorithm is required to be overly specific, with respect to the input examples in each case. Suppose that the input to the specialization algorithm could have overly specific aspects. Then it would not be possible to determine whether a failing positive example was a result of removing a candidate rule for specialization from the theory or the result of one of the existing specialization errors in the theory (that is, an overly specialized rule or a missing rule). If the same example showed up as both a positive and negative example for a rule, this might not indicate an overlap problem at all, but merely the interaction of specialization and generalization errors. The same kind of arguments hold for the generalization algorithm.

With reference to a given correction algorithm (generalization or specialization), let us define *complementary* examples to be examples that are incorrectly classified by the initial theory but that are not the focus of the algorithm (that is, failing negatives in the case of theory generalization, and failing positives in the case of theory specialization). Then the above considerations lead to the following constraint on the correction algorithms:

the input to each correction algorithm must not contain complementary examples.

Satisfying this constraint requires that no failing negative examples be input to the generalization algorithm and no failing positive examples be input to the specialization algorithm.

In practice, this is accomplished in EITHER by removing the failing negative examples from the input to the generalization algorithm and using the output of the generalization algorithm as input to the specialization algorithm.

4.2 Theory-based Response

Fortunately, there is a simple solution to the correctability problem in the case of noise-free data. (See Chapter 6 for a discussion of the complications introduced by noise.) In the worst case, a cover can be selected consisting entirely of category rules (rules whose consequent is a category). Since these rules imply a single category, updates to them cannot affect membership in other categories. For example, if a given example is erroneously provable as a member of given category, then, for example, specializing the antecedents of the corresponding category rule will cause the example not to be provable in that category without possibly affecting membership in other categories. In general, these rules can be modified using the correct set of positive and negative examples (that is, examples that are incorrectly proven in the given category or which fail to be proven in the category) resulting in an updated theory that is consistent with the training set.

However, we would prefer *not* to make the corrections at the root level of the theory. This is because strengthening lower level rules allows them to participate in proofs involving more than one category, thereby strengthening the theory as a whole, rather than the portion that is devoted to a single category (see Chapter 5). For example, suppose the theory for cups had an additional (nonsensical) *liftable* rule that stated that all things made out of aluminum were liftable. Clearly, the top level cup rule could be replaced by a set of rules in which the *liftable* antecedent was replaced by constraints that distinguished between the positive and negative examples for cup (that is, those that were liftable according to the correct definition of liftability, and those that were aluminum but were not otherwise liftable). This correction would have two undesirable effects. First, for the cup category, we would have lost the benefit of the correct portion of the liftable rules and replaced them with a set of rules that were merely consistent with the training examples. Second, any other categories that used the liftable rules would not have had the benefit of the correction.

Because of these considerations, EITHER initially starts with rules that are located at the leaves of the proof tree (that is, rules involving observables), and only modifies higher level rules if necessary. If the leaf cover has no overlapping examples, then no changes to intermediate rules are required. If, however, there are rules with overlapping examples, EITHER replaces each such rule with its parent rules, and tests the parents for overlapping examples. This process continues until a set of rules is obtained that introduces no overlapping examples, and these rules are used to form the final correction.

4.3 Cover-based Response

The approach shown in the previous section will always result in a consistent theory, when noise-free data are input. However, in some cases it is possible to solve the correctability problem through proper selection of the cover that is input to the correction algorithms. This is desirable, since the input cover consists of leaf-level rules, which increases the possibility of shared rules.

Recall that there are two kinds of covers used by EITHER: the antecedent cover and the rule cover. The antecedent cover is used in the generalization algorithm and consists of a set of rules taken from the various partial proofs of the failing positive examples. It is constructed such that if all of the specified antecedents were retracted from each of the

rules, at least one proof of each example would be completed. Clearly, there are potentially many such covers for a typical theory. The minimum rule cover is used by the specialization algorithm. It consists of a set of rules taken from the proofs of the failing negative examples, such that if each rule in the cover is retracted from the theory none of the failing negative examples will be provable. Here also there are potentially many such covers.

In the implemented version of EITHER, cover correctness processing is only performed on the rule cover, for reasons of computational efficiency (see Section 4.4). In the generalization algorithm, when there are conflicting examples rule corrections are always biased in favor of the failing positive examples so that the updated theory is never overly specialized.

In the case of theory specialization, EITHER has two methods for finding the actual cover that is submitted to the correction algorithm. The first is the *greedy algorithm*. This method recursively selects rules that are used in the most proofs to add to the cover, stopping when all of the proofs have been covered. The other algorithm, called the *exhaustive cover algorithm*, evaluates each cover for correctness and selects the cover with the minimum number of overlapping examples. If there are multiple covers, each of which have no overlapping examples, the exhaustive cover algorithm chooses the cover using a LEF whose criteria are minimum cover length and minimum number of failing positive examples when the cover is removed from the theory.

To illustrate the actions of the rule cover algorithm in the presence of correctness problems, consider the following theory:

$$\begin{aligned} r_1 : c_1 &\leftarrow x \wedge y \\ r_2 : c_2 &\leftarrow x \wedge z \\ r_3 : x &\leftarrow a \\ r_4 : y &\leftarrow b \\ r_5 : z &\leftarrow b \end{aligned}$$

Assume that the rules r_4 and r_5 are erroneous and that their correct form should be:

$$\begin{aligned} r_4 : y &\leftarrow b \wedge s \\ r_5 : z &\leftarrow b \wedge w, \end{aligned}$$

where a , b , s , and w are observables. Suppose that we have two examples, one of category c_1 , and one of category c_2 , called examples 1 and 2, respectively. Let the observables for these examples be:

1. $a \wedge b \wedge s$
2. $a \wedge b \wedge w$.

This is an example of an overly general theory, since each example would be provable in the other's category, as well as its own. The rule covers for these examples are:

$$r_3, r_3 \wedge r_4, r_3 \wedge r_5, r_3 \wedge r_4 \wedge r_5, \text{ and } r_4 \wedge r_5.$$

If we retract r_3 , then both examples will fail as examples of their own category. Hence, both examples are the *failing positive examples* for r_3 , examples that fail to be proven in their categories when r_3 is retracted. However, both examples are also *failing negative examples* for r_3 , examples that are provable in another category and which require r_3 for the incorrect proof. As a result, no inductive specialization of r_3 can be formed that correctly distinguishes between the two examples.

However, if the cover $r_4 \wedge r_5$ is chosen, new rules for r_4 and r_5 that do correctly distinguish between the two examples can be obtained. These rules can be derived as follows: the negative example for r_4 is example 2 (that is, it is fallaciously provable as a category c_1 example and requires r_4 for its proof). The positive example for r_4 is example 1 (that is, when r_4 is removed from the theory, example 1 fails to be provable in its own category,

which is c_1). Using these two examples will cause the additional antecedent, s , to be added to the antecedents of r_4 , resulting in the correct rule.

A similar analysis shows that using the cover $r_4 \wedge r_5$ also results in the correct modification to r_5 .

The trace below shows EITHER's response to this theory when the exhaustive cover algorithm is used.

There are provable negative examples so will specialize theory.
 Finding minimum rule cover.
 These were reduced to 2 final covers from which to find minimum.
 Cover 1 results: Minimum overlapping examples 2,
 minimum disprovable positive 2, minimum cover length 5.
 Cover 2 results: Minimum overlapping examples 0,
 minimum disprovable positive 2, minimum cover length 10.
 Rule cover complete.
 Starting correctability check on initial cover.
 Rule
 $((z) \leftarrow (b))$ was correctable
 so am making correction to the theory.
 Removing rule
 $((z) \leftarrow (b))$ caused positive examples to fail,
 so will form new rules.
 Adding rules $((\leftarrow (z) (b) (\text{not } (s))))$ to theory.
 Rule
 $((y) \leftarrow (b))$ was correctable
 so am making correction to the theory.
 Removing rule
 $((y) \leftarrow (b))$ caused positive examples to fail,
 so will form new rules.
 Adding rules $((\leftarrow (y) (b) (s))))$ to theory.
 Correctability check complete.

Syntactical simplicity check omitted from trace.
--

Initial rule:
 $((z) \leftarrow (b))$
 updated version
 $((z) \leftarrow (b) (\text{not } (s)))$
 Initial rule:
 $((y) \leftarrow (b))$
 updated version
 $((y) \leftarrow (b) (s))$
 Beginning rule reduction process.
 Rule reduction complete.
 (
 $((c1) \leftarrow (x) (y))$
 $((c2) \leftarrow (x) (z))$
 $((x) \leftarrow (a))$
 $((z) \leftarrow (b) (\text{not } (s)))$
 $((y) \leftarrow (b) (s))$
)

Note that EITHER only appended (s) and $(\text{not } (s))$ to the rules for (y) and (z) , since this was sufficient to discriminate between the examples.

When the greedy algorithm is used to form the cover, the following trace results:

There are provable negative examples so will specialize theory.
 Finding minimum rule cover.
 2 proofs remain to be covered.
 0 proofs remain to be covered.
 Rule cover complete.

The greedy algorithm selects the rule $((x) \leftarrow (a))$ as its initial cover, which has overlapping examples.

Starting correctability check on initial cover.
 There are overlapping examples for
 rule
 $((x) \leftarrow (a))$, so will replace with parent rules,
 $((c2) \leftarrow (x) (z))$
 $((c1) \leftarrow (x) (y))$
 Rule
 $((c2) \leftarrow (x) (z))$ was correctable
 so am making correction to the theory.
 Removing rule
 $((c2) \leftarrow (x) (z))$ caused positive examples to fail,
 so will form new rules.
 Adding rules $((\leftarrow (c2) (x) (z) (not (s))))$ to theory.
 Rule
 $((c1) \leftarrow (x) (y))$ was correctable
 so am making correction to the theory.
 Removing rule
 $((c1) \leftarrow (x) (y))$ caused positive examples to fail,
 so will form new rules.
 Adding rules $((\leftarrow (c1) (x) (y) (s))))$ to theory.
 Completed corrections to parents of rule
 $((x) \leftarrow (a))$
 Correctability check complete.

Syntactical simplicity check omitted from trace.

Beginning rule reduction process.
 Rule reduction complete.

```
((x) <- (a))
((y) <- (b))
((z) <- (b))
((c2) <- (x) (z) (not (s)))
((c1) <- (x) (y) (s)))
```

The final correction that EITHER makes is consistent with the examples, but it is made to the category rules rather than the leaf-level rules.

4.4 Implementation Simplifications

In order to attain efficiency advantages, the implemented EITHER algorithm includes some changes from the ideal algorithm described above in the case of multiple category learning.

Firstly, we do not consider overlapping examples when selecting the minimum antecedent cover. The effect of this is that the theory formed by the generalization algorithm

may be inconsistent with the provided training examples in that it may introduce new failing negative examples. However, these examples will be included with the original failing negative examples from the original incorrect theory as input to the specialization algorithm, so that the final theory produced by EITHER *will* be consistent with both the failing negative and failing positive examples. This guarantee is possible because the implemented specialization algorithm *does* select a rule cover with no overlapping examples, and the output from the generalization algorithm is guaranteed to have no failing positive examples.

Computationally, the benefits of this simplification are that, each time the antecedents for a rule are removed, EITHER does not have to consider all of the input training examples to see if any of them become provable in other categories. If this was done, EITHER would have to consider all possible proofs of each example in all other categories to insure that no overlapping examples were obtained. These proofs would have to be considered for all of the candidate covers corresponding to the partial proofs of the examples, rather than simply selecting the minimum length cover, as is now done.

Instead, EITHER adds each failing negative example so introduced as input to the specialization algorithm. These examples may cause the minimum rule cover algorithm to add new terms to the set of unique covers returned, based on the rules used in the proofs of the examples. To insure correctability, we only need to check that, when a rule in a candidate cover is removed, each example remains provable in its *own* category (that is, we do not have to check all other categories). The only additional cost here is the added cost associated with the additional rules added to the covers. (In general this will be the generalized rules that caused the correctability problems in the first place.) A further optimization is to consider for overlap only those examples that fail to be proven in their category when the cover as a whole is removed.

Finally, as a run time optimization, a switch is provided to cause the program to select the first correctable rule cover that it obtains, rather than continuing to search for the minimum length correctable cover, as is done by the ideal algorithm. Although in the worst case this still might require that all of the candidate covers be evaluated (when the only correctable cover is the last one among the candidate covers), in practice usually far better performance is achieved, since the theories that EITHER has been applied to seem to have a fairly high density of correctable covers.

4.5 Experimental Results

In order to demonstrate EITHER's capability when given a multiple category theory as input, EITHER was tested against the *Soybean* theory, presented in [Michalski and Chilausky, 1980]. This is a theory for diagnosing soybean diseases that distinguishes between nineteen possible soybean diseases using examples that are defined in terms of thirty five features. The details of the soybean theory are presented in Section D.1. Summary information for the DNA theory is given below.

Number of rules: 73
 Number of consequents: 37
 Number of symbols: 325
 Average number of disjuncts: 1.97
 Average number of antecedents: 3.45

The original theory associated probabilistic weights with certain disease symptoms. In addition, some groups of disease symptoms were regarded as *significant* while other groups were regarded as confirmatory. The theory was translated to propositional Horn

clause format³ by only including the significant symptoms and by deleting any symptom from the theory that had a weight less than .8.

Unfortunately, the classification performance of the Horn clause formulation appeared to be seriously deficient, when compared to the probabilistic representation, since the probabilistic formulation appeared to more accurately mirror the knowledge associated with the domain. For example, the propositional Horn clause version of the theory obtained a 12.3% classification performance compared to 73% for the probabilistic version.

To circumvent the problem, a "fuzzy" tester⁴ was used to classify the test examples, based on the updated theory provided by EITHER. The fuzzy tester algorithm accounts for two possible classification problems with the EITHER-generated theory. The first problem occurs when a test example is provable in more than one category. This indicates that the theory generated by EITHER is overly general with respect to the test examples. The second problem occurs when a test example is not provable in any category by the revised theory. This problem corresponds to a theory that is overly specific with respect to the test examples. With the standard test logic implemented for EITHER, the example would be assigned to the most common category in the training examples in either case.

In contrast, the reasoning used in the original soybean tests would assign a weight to each possible category associated with the example, and choose the category with the highest weight. The fuzzy tester used by EITHER is a simple approximation that operates differently for overly general and overly specific theories. For an overly general theory, the tester selects the category that makes the most use of the example features. This is done by choosing the category whose proof of category membership has the greatest number of example features used in the proof. In the case of an overly specific theory, the fuzzy tester chooses the category that comes nearest to being provable for the theory. This is done by choosing the category with a *partial* proof of category membership that has the least number of conflicting antecedents.

The performance results for the soybean experiments are shown in Figure 4.2. In each test, performance was measured against seventy five test examples. The number of training examples was varied from one to one hundred, with the training and test examples drawn from the entire example population, with no overlap. The results were averaged over 21 samples. Note that even with the fuzzy tester logic, EITHER's initial performance was only 51%, as compared to 73% for the original results presented in [Michalski and Chilausky, 1980]. However, the figure makes clear that EITHER maintains its initial performance advantage over the entire training interval. In fact, a one-tailed Student t-test on paired differences showed that the superior performance of EITHER is statistically significant at at least the 5% level for every point plotted on the learning curves.

Section E.2 presents a typical example of the type of theory modifications made by EITHER during the course of the soybean tests. The modifications include: removing antecedents, generalizing antecedents, inductively creating new rules, popping up due to the removal of all the antecedents from a particular rule. Both rule specialization and generalization were required to correct the theory. The final rules were compressed using both the inter and intra operators. A new concept, intra-2641, that represents a subset of the values for the crop_history attribute, was discovered. Intra-2641 was found to be useful for both generalization and specialization.

³By Jeff Mahoney, in connection with a separate project.

⁴The fuzzy tester was also developed by Jeff Mahoney as part of his research project.

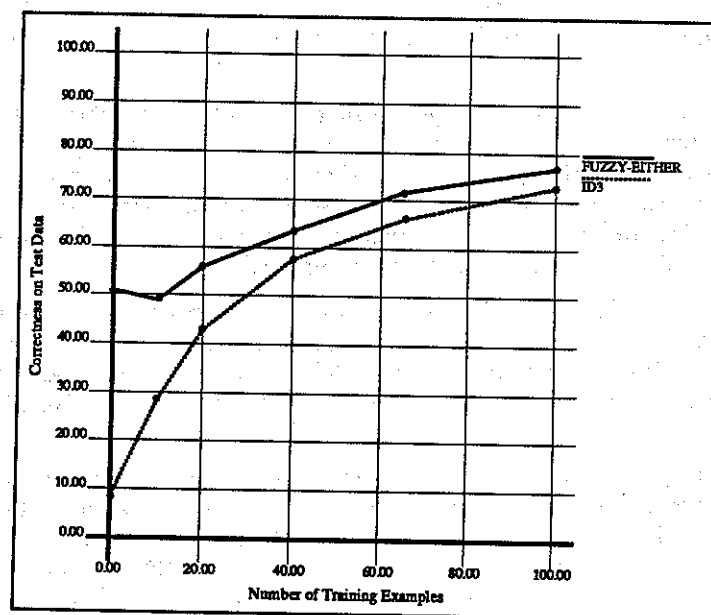


Figure 4.2: EITHER Performance Results for the Soybean Theory

Chapter 5

Improving Shared Rules

Frequently, domain theories involve multiple concepts that share intermediate rules. For example, in a theory for a variety of animal species, the concepts for giraffes and tigers might share knowledge about mammals, and the concepts for penguin and duck might share knowledge about birds. Improving rules for the shared concept "mammal" can increase classification accuracy for both giraffes and tigers. However, refining rules for such shared intermediate rules must be done carefully in order to account for the impact on all of the concepts that use them.

Revising rules for shared concepts frequently has the interesting effect of improving performance on test data that is drawn from a completely different population from the training data. For example, if the system is trained on ostriches, it may improve performance when tested on penguins. This effect is called *cross-category transfer*. Here, cross-category transfer occurs when learning about ostriches causes the system to revise its shared knowledge about birds, for example by learning about non-flying birds.

Standard empirical learning systems are clearly incapable of exhibiting cross-category transfer. Most theoretical work in learning also assumes that the distribution of examples is the same during training and test [Valiant, 1984]. The unique properties of revising multi-concept theories with shared rules allows for transfer to diverse testing environments.

EITHER responds to failing negative examples by performing rule specialization. Similarly, the response to failing positive examples is rule generalization. EITHER determines the proper location for the rule correction based on a syntactic simplicity measure (Section 3.1), subject to a correctability constraint (Chapter 4). EITHER analyzes the existing theory to determine if the antecedents of the corrected rule can be connected to existing rules in the theory, attempting to maximize the use of the existing theory (Section 3.2.1). When rules are inductively learned, EITHER looks for structure in the rule updates and uses this to compress the resulting change to the theory (Section 3.2.2). The resulting theory update approach tends to focus corrections on rules that are used by multiple categories, as will be explained below.

5.1 Analysis

The syntactic simplicity check, described in Section 3.1, has the beneficial side effect of tending to cause EITHER to make rule corrections at or below the level of the first shared rule. Two examples will be presented to illustrate this effect. In the first case, we consider a rule structure in which a higher level rule is supported by several rules at the next lower level:

$$\begin{array}{l} w \leftarrow x \\ x \leftarrow a \end{array}$$

$$\begin{aligned}x &\leftarrow b \\x &\leftarrow c.\end{aligned}$$

Assume that the first rule should correctly be:

$$w \leftarrow x \wedge d \wedge e.$$

If EITHER was given examples based on the correct theory, and was forced to make the corrections to the theory at the leaf level, then EITHER would obtain the corrected theory (that would be consistent with the input examples):

$$\begin{aligned}w &\leftarrow x \\x &\leftarrow a \wedge d \wedge e \\x &\leftarrow b \wedge d \wedge e \\x &\leftarrow c \wedge d \wedge e.\end{aligned}$$

Clearly, changing the theory at the higher level would result in a much simpler syntactic change to the theory. Assuming the leaf level rules were in error would also mean that the original author of the system would have had to make the same error in three separate rules whereas only one error would be necessary for the higher level rule to be incorrect.

As a second example, consider a theory in which a single rule supports multiple parents:

$$\begin{aligned}w &\leftarrow x \\y &\leftarrow x \\z &\leftarrow x \\x &\leftarrow a,\end{aligned}$$

and assume that the correction to the theory should be:

$$x \leftarrow a \wedge b \wedge c.$$

If the correction is made at the higher level, then each of the rules "w" "y" and "z" would have to have the antecedents *b* and *c* added, which would be clearly more complicated syntactically. As a result, EITHER tends not to make a correction in the theory at a level above the first shared rule.

In fact, the implemented version of EITHER enforces a stronger constraint that prevents it from rising above shared rules. The implemented syntactic simplicity check groups the rules at the current level into sets having a single parent. It then compares the correction to the grouped rules against the correction to their parent. The parent rule will be substituted for the grouped rules whenever its correction is simpler. If a particular rule has multiple parents, the syntactic simplicity check is not made, resulting in no corrections above the level of the first shared rule.

The effect of this process is to ascend to higher level rules whenever their correction is simpler, but to make the corrections at or below the level of the first shared rule (a rule having multiple parents). The only exception occurs when the shared rule suffers from correctability problems. As a result, EITHER tends to maximize the effect of shared rules in the theory.

5.2 Experimental Results

EITHER has been tested on several multi-category domain theories with shared intermediate rules. This section presents results showing that EITHER can successfully revise shared rules and that such revisions result in positive transfer to test data that are very different from the training data.

Artificial data were automatically generated for the animal theory shown in Figure 5.1 (originally presented in Chapter 3 and repeated here for convenience), and a similar theory shown in Figure 5.2 for classifying different types of computers based on their appearance.

Imperfect versions of the animal and computer theories were also constructed. In each case the rules for some of the intermediate concepts were corrupted in order to allow for cross-category transfer. The corrupted version of the mammal and bird rules is shown in Figure 5.3 and the corrupted version of the micro, unix-workstation, and lisp-machine rules are shown in Figure 5.4. The faults introduced include missing rules, additional antecedents, and missing antecedents. These theories were given to EITHER to revise.

In order to illustrate cross-category transfer, the system was trained on examples from some of the categories and tested on examples of the remaining categories. In the animal domain, the system was trained on giraffes, cheetahs, dolphins, bats, platypuses, ostriches, and ducks. It was tested on zebras, tigers, whales, penguins, and grackles. In the computer domain, EITHER was trained on Macs, Suns, and Explorers. It was tested on MacII's, HP's, and Symbolics. Learning curves were generated by performing batch training on increasingly larger fractions of a set of training examples and by repeatedly testing predictive accuracy on the same disjoint test set. The final results were averaged over 20 random selections of training and test sets.

The results are shown in Figure 5.5. With training, EITHER improves its performance on the test data despite the fact that the training and test data are drawn from completely different populations. The improvement occurs because EITHER revises rules for shared intermediate concepts. For example, when trained on ostriches and dolphins it learns that birds do not have to fly (they just need to have feathers) and that mammals do not have to possess legs. These revisions allow it to classify penguins and whales more accurately during testing. Note, however, that the revision is not perfect: in the animal domain EITHER's performance levels off at about 85%. This is because some of the theory errors in the animal theory case were to *category* rules (duck, ostrich, penguin). In the case of the penguin rule, a generalization was required. Because the system never saw any examples of penguins during training, this generalization was not possible, and the penguin rule was not corrected. Notice, however, that a normal inductive learning system (for example, ID3 [Quinlan, 1986a]) would remain at 0% accuracy in this situation. Since ID3 never sees any examples of penguins or whales in the training set, it will never classify a test example as a penguin or a whale.

Figure 5.6 shows EITHER's performance in standard learning, where the training and test populations are the same. Note that EITHER quickly achieves high performance compared to a standard inductive learning (ID3), and maintains a significant advantage as the number of training examples increases. In most cases, EITHER converged to the exact theory after 65 examples for the animal theory and after 35 examples for the computer theory.

(mammal)	←	(body-covering hair)
(mammal)	←	(feed-young milk)
(mammal)	←	(birth live)
(bird)	←	(body-covering feathers)
(bird)	←	(birth egg) (fly)
(ungulate)	←	(mammal) (foot-type hoof)
(ungulate)	←	(mammal) (ruminant)
(carnivore)	←	(eat-meat)
(carnivore)	←	(teeth pointed) (foot-type clawed)
(giraffe)	←	(ungulate) (neck-length ?n) ($\geq ?n 5$) ($\leq ?n 6$) (color tawny) (pattern spots) (pattern-color black)
(zebra)	←	(ungulate) (color white) (pattern stripes) (pattern-color black)
(cheetah)	←	(mammal) (carnivore) (color tawny) (pattern spots) (pattern-color black)
(tiger)	←	(mammal) (carnivore) (color tawny) (pattern stripes) (pattern-color black)
(dolphin)	←	(mammal) (fore-appendage fin) (color gray) (body-covering moist-skin) (body-length ?b) ($\geq ?b 4$) ($\leq ?b 6$)
(whale)	←	(mammal) (fore-appendage fin) (color gray) (body-covering moist-skin) (body-length ?b) ($\geq ?b 10$) ($\leq ?b 60$)
(bat)	←	(mammal) (color black) (pattern none) (pattern-color none) (fly)
(platypus)	←	(mammal) (birth egg) (foot-type webbed)
(ostrich)	←	(bird) (not (fly)) (neck-length ?n) ($\geq ?n 3$) ($\leq ?n 4$) (color white) (pattern patch) (pattern-color black)
(penguin)	←	(bird) (color white) (pattern patch) (pattern-color black) (foot-type webbed) (not (fly))
(duck)	←	(bird) (foot-type webbed) (fly)
(grackle)	←	(bird) (color black) (pattern none) (pattern-color none) (fly)

Observable Features: feed-young body-covering birth eat-meat fly teeth fore-appendage foot-type neck-length body-length color pattern pattern-color ruminant

Categories: giraffe zebra cheetah tiger dolphin whale bat platypus penguin ostrich duck grackle

Figure 5.1: Animal Domain Theory

(workstation)	←	(screen-size ?s) (\geq ?s 14)
(workstation)	←	(not (floppy))
(micro)	←	(floppy)
(micro)	←	(screen-size ?s) (\leq ?s 10)
(lisp-machine)	←	(workstation)(meta-key)
(unix-workstation)	←	(workstation)(not (meta-key))
(macintosh)	←	(micro) (desktop-display)
(macintosh)	←	(micro) (mouse mechanical) (mouse-buttons 1)
(pc)	←	(micro) (mouse none) (mouse-buttons none)
		(not (desktop-display))
(mac)	←	(macintosh) (box-width ?b) (\leq ?b 11)
(macII)	←	(macintosh) (box-width ?b) (\geq ?b 17)
(sun)	←	(unix-workstation) (mouse optical)
		(mouse-buttons 3) (box-height ?h) (\leq ?h 12)
(hp)	←	(unix-workstation) (mouse mechanical)
(explorer)	←	(lisp-machine) (mouse optical) (mouse-buttons 3)
		(keyboard-width ?k) (\geq ?k 18)
(symbolics)	←	(lisp-machine) (mouse mechanical)

Observable features: floppy floppy-size mouse mouse-buttons windows desktop-display meta-key screen-size box-width box-height keyboard-width
Categories: pc mac macII sun hp explorer symbolics

Figure 5.2: Computer Domain Theory

Correct Rules		Corrupted Rules	
(mammal)	← (body-covering hair)	(mammal)	← (body-covering hair)
(mammal)	← (feed-young milk)	(mammal)	← (fore-appendage leg)
(mammal)	← (birth live)	(mammal)	← (feed-young milk)
		(mammal)	← (fore-appendage leg)
(bird)	← (body-covering feathers)	(mammal)	← (birth live)
(bird)	← (birth egg) (fly)		← (fore-appendage leg)
(duck)	← (bird)		*retracted*
	(foot-type webbed) (fly)	(bird)	← (fly)
(ostrich)	← ... (not (fly))	(duck)	← (bird)
(penguin)	← ... (not (fly))		(foot-type webbed)
		(ostrich)	← ...
		(penguin)	← ...

Figure 5.3: Corrupted Animal Theory

Correct Rules		Corrupted Rules	
(micro)	← (floppy)	*retracted*	
(lisp-machine)	← (workstation)	(lisp-machine)	←
	(meta-key)	(workstation)	
(unix-workstation)	← (workstation)	(unix-workstation)	←
	(not (meta-key))	(workstation)	

Figure 5.4: Corrupted Computer Theory

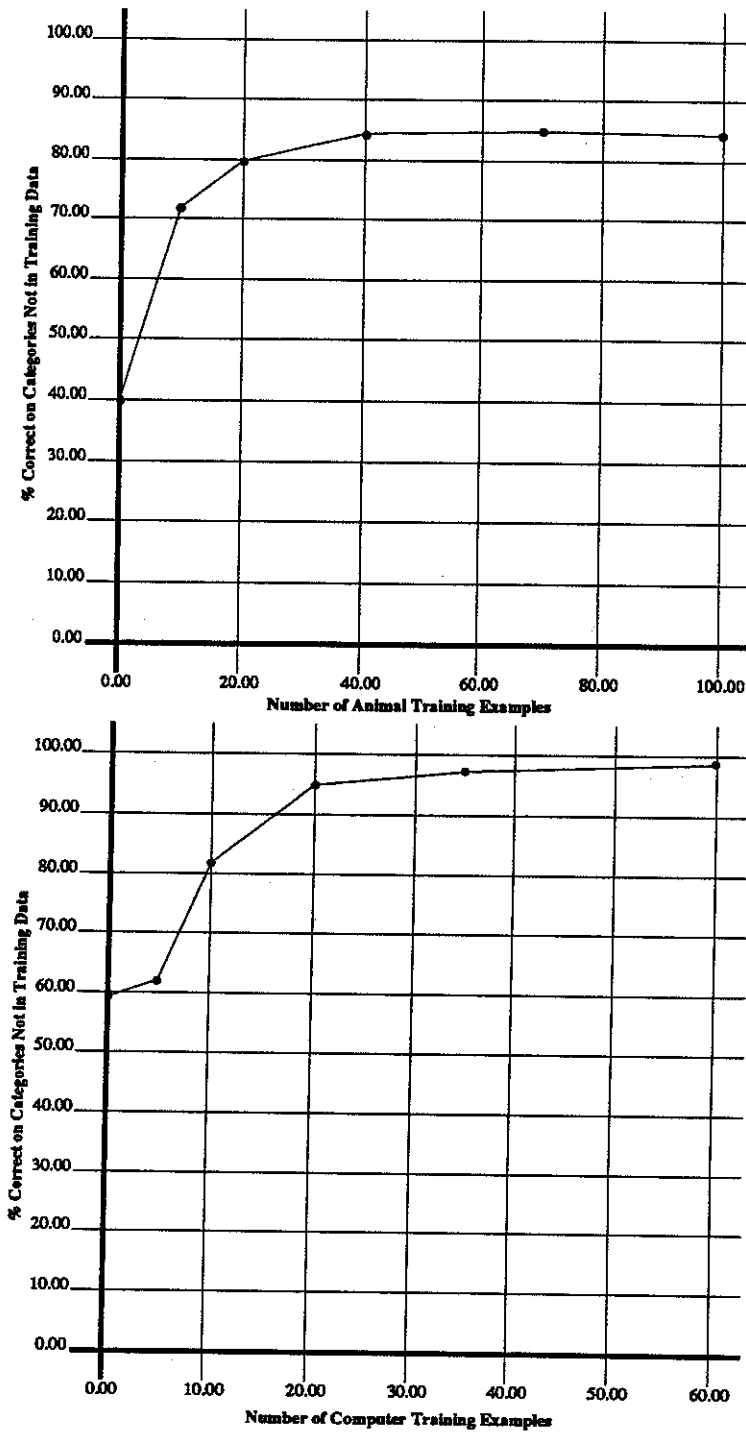


Figure 5.5: Cross-category Learning in the Animal and Computer Domains

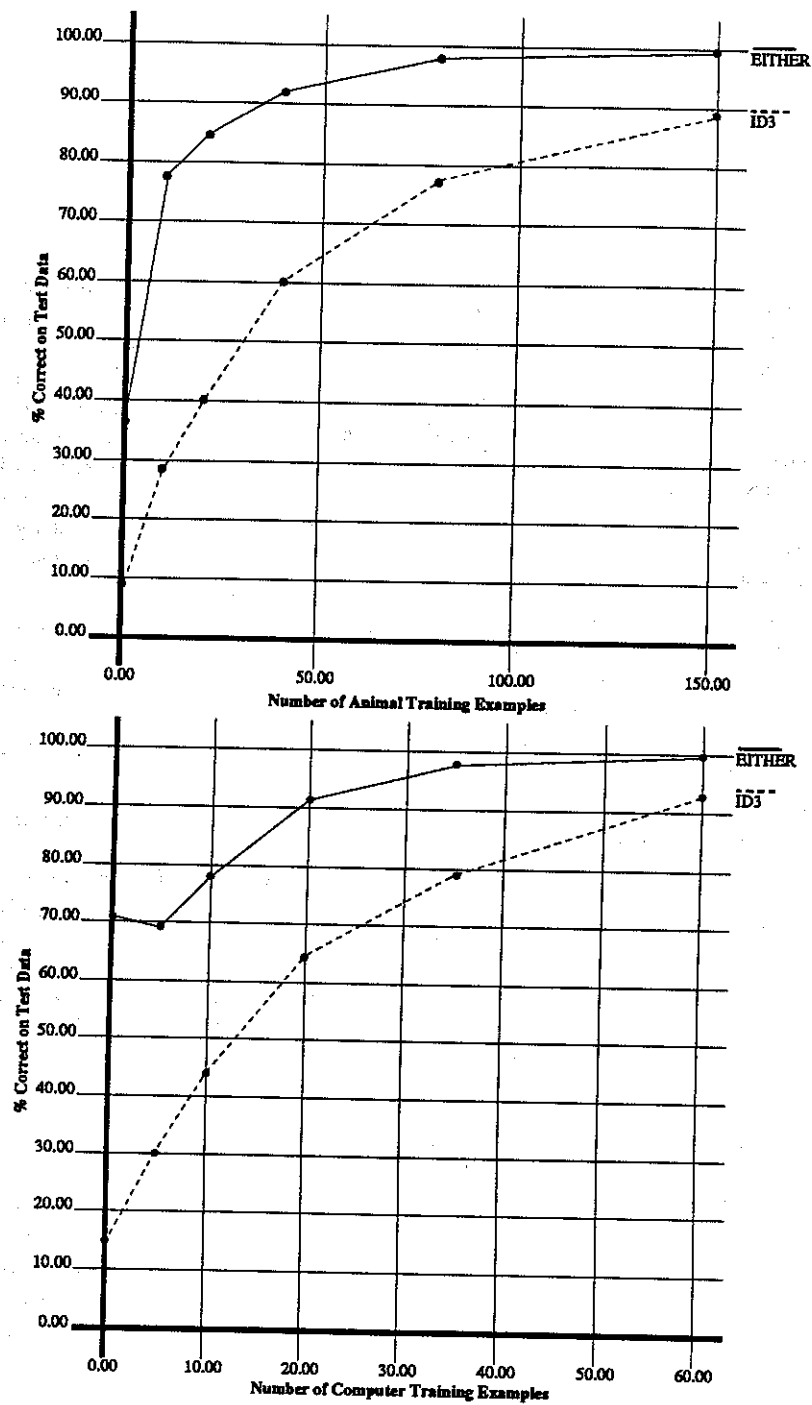


Figure 5.6: Standard Learning in Animal and Computer Domains

Chapter 6

Response to Noise

For examples consisting of attribute-value lists, there are two primary sources of noise. The first is *mis-specification*: the value for an attribute or category may be incorrect for a variety of reasons, including typographical errors, errors in measurement, and perception errors. The second type of noise is called *residual variation* [Mingers, 1989]. It refers to additional factors that affect the results but that are not recorded. Residual variation is a fairly common source of noise in real-world applications. It can occur because those recording the data were either unaware of the affect of the additional factors, or they were simply unable to record them. With either source of noise, one effect is examples that are inconsistent (that is, two examples that have identical attributes but are labeled in different categories). A second effect is examples with features that should correlate with their categories, but which do not because of noise masking the correlation.

One of the main problems with learning algorithms that do not account for noise is that they may tend to “overfit” the data. That is, elaborate rules or decision structures are generated which serve to account for a small number of anomalous examples among the training set. Generating the additional rules not only increases the processing time, but in general will actually decrease performance since the learning algorithm is learning rules that only apply to the noisy examples and may mis-identify correct examples.

Response to noisy data shows up at three points in the EITHER processing: in the generation of the minimum cover, in the one-sided corrections made by EITHER that do not require the inductive learner, and in the rule corrections supplied by the inductive learner. When noisy data are submitted to EITHER, the resulting theory corrections can no longer be guaranteed to be consistent with all of the training data. (As mentioned above, examples in the training data may not be consistent with each other.) When noise processing is invoked in EITHER, the corrected theory may not even be consistent with *noise-free* training data. The next three sections describe EITHER’s response to noise in more detail. The final section of this chapter shows performance results for the noise processing algorithms.

6.1 Accounting for Noise in the Minimum Cover

The principle behind handling noise in cover generation is that there is more confidence associated with corrections to rules that apply to several failing examples. If the correction only applies to a single example, then it could well be that it is the *example*, rather than the rule that is in error. However, if the same (erroneous) rule is used by several failing examples, it is unlikely that all of the examples happen to have noisy attributes that cause them all to focus on the same rule.

To be specific, for failing positive examples, focusing on the same rule would require that all of the partial proofs of the failing positives happen to use the same rule. In each case the same set of antecedents for the rule would have to conflict with the features of the example. This is unlikely to happen if the examples are subject to random feature

noise. For failing negative examples, this would mean that all of the incorrect proofs of the negative examples use the same rule. This is unlikely to happen if the incorrect proofs of the examples are due to random feature noise in the examples.

EITHER's response to noise in these cases is to prune the minimum covers provided to the rule correction algorithms. For antecedent covers, failing antecedents are grouped into separate sets as necessary to account for the failing positive examples associated with each rule. That is, a new antecedent set is created whenever the conflicting antecedents for a particular example are different from any of the antecedent sets already associated with the rule. When the cover is completed, noise processing removes those antecedent sets from the cover that correspond to only a single example. In minimum rule cover processing, terms in the cover that only correspond to a single example are also removed. In this case, any rule in the cover that only corresponds to a single failing negative is removed.

Because EITHER is now using a partial cover, the corrected theory will not in general be consistent with the training set. However, the benefit of the partial cover in responding to noise should actually improve as the number of training examples increases. This improvement stems from the decrease in probability that a bad rule will only affect a single example as the number of examples increases. Since EITHER attempts to select rules that correspond to multiple examples, the probability of selecting a spuriously correlated rule will also decrease.

6.2 Accounting for Noise in One-Sided Corrections

EITHER also provides a response to noisy data in the cases where a one-sided correction is attempted. (That is, a correction based on either failing positive examples or failing negative examples, but not both.) These corrections occur in the following cases:

- When a rule is generalized by retracting an antecedent, and no negative examples become provable using the generalized rule. In this case the final correction would be to remove the antecedent from the rule.
- When the theory is specialized by removing a rule from the theory, and all positive examples remain provable. In this case the most drastic correction to the theory would be to delete the rule. However, EITHER may instead specialize the antecedents to the rule when it lacks the confidence necessary to justify rule deletion.
- When it is possible to generalize the rule by generalizing the antecedent that fails in the input version of the rule. Here, if possible, EITHER generalizes the antecedent such that it covers all of the failing positive examples associated with the rule and does not cover any of the negative examples.

To account for noise, EITHER pre-processes the input data prior to entering the calculations described above. The pre-processing removes inconsistent examples in the following way:

- EITHER checks the input examples for inconsistency. Inconsistency is discovered when two examples with differing category labels are found to have the same attribute values.
- For each set of inconsistent examples, if a majority belong to a particular category, EITHER deletes the examples in other categories from the input data.

This pre-processing causes EITHER to ignore examples with obvious noise problems and provide corrections that are consistent with the remaining examples.

6.3 Accounting for Noise in Inductive Rule Formation

EITHER uses ID3 as its inductive component. Quinlan has developed a method for dealing with noisy data in ID3 [Quinlan, 1986b]. In this method, when an attribute is being considered as a possible test feature in the decision tree, the values for the attribute are checked to see if they are independent of the example categories using a chi-squared test. If so, this indicates that the attribute values may include noise and will not be predictive of example category membership. When all of the remaining attributes fail the chi-squared test, the current node is installed as a leaf node in the decision tree (that is, the decision tree is "pruned" of all possible subtrees that could have originated from this node). The category for the node will be the majority category among the training examples input to the node.

The decision trees formed using this approach can no longer guarantee consistency with the training examples. However, it has been shown that the pruned trees can provide improved accuracy on the test data [Mingers, 1989; Quinlan, 1986b]. The trees are also simpler in that they do not include the complex subtrees necessary to account for the noisy training examples.

The chi-squared noise processing algorithm has been incorporated in the version of ID3 used by EITHER. Incorporating this algorithm means that the examples sent to ID3 will be checked for noise, and the decision trees will be pruned if necessary. Since EITHER transforms these decision trees directly into rules, the resulting rules should be syntactically simpler and should show improved performance over the test data. However, when noise processing is invoked in the inductive component, EITHER can no longer guarantee consistency with the training examples.

6.4 Noise Performance Results

The changes to EITHER described above were tested on revising domain theories using data artificially corrupted with noise. Feature and category noise were added to the training examples, and feature noise only to the test examples. Feature noise was added by substituting a random value from the feature's domain. Category noise was added to the training examples by substituting a random category. Category noise was *not* added to the test examples in order to identify how well EITHER was able to identify the correct categories for the test examples. The noise level identifies the frequency of noisy data: a noise level of 0.1 means that 10% of the features and categories were randomly corrupted. Data were automatically generated for the animal and computer theories described in section 5.2. The data for the theories were generated by first generating noise-free examples as described in section 5.2 and then adding noise as described above. Noise performance was measured using corrupted versions of these theories that had the faults described in Section 5.2.

Figure 6.1 shows the degradation in predictive accuracy with increasing levels of noise for EITHER with cover truncation (EITHER-PRUNE), EITHER without cover truncation (EITHER-FULL), and ID3 (with chi-squared pruning). Since EITHER uses ID3 as its inductive component, ID3 is the same as EITHER when it is not given an initial theory. The animal results are averaged over 8 trials of 60 training and 100 test examples. The computer results are averaged over 23 trials of 40 training and 100 test examples. Due to its initial theory, EITHER-PRUNE always performs better than ID3. A statistical t-test shows that the difference is significant at each noise-level plotted except 0.2. With little or no noise (noise levels of 0.0 and 0.01) both versions of EITHER perform approximately the same. However, as the noise level is increased, the effect of pruning becomes more pronounced. The difference is statistically significant for levels of noise greater than 0.01. At noise levels of 0.1 and 0.2, pruning provides approximately a 9% advantage on animals and 4% advantage on computers. Unlike EITHER-PRUNE, EITHER-FULL eventually performs worse than ID3 at

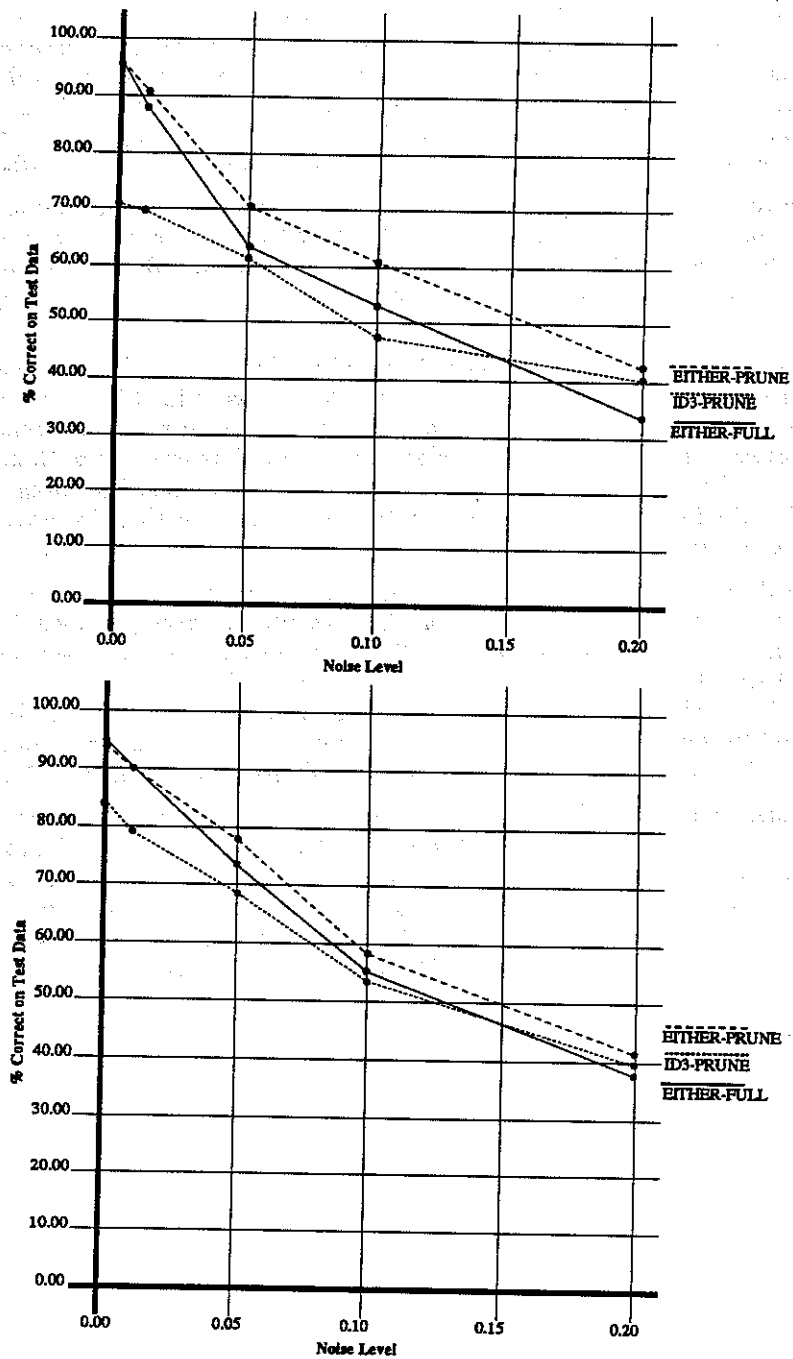


Figure 6.1: Test Accuracy vs. Noise Level for Animal and Computer Domains

a noise-level of 0.2. Of course, one would expect the accuracy of all systems to converge to random chance as noise-level is increased to 1.0.

Unlike EITHER-FULL, EITHER-PRUNE's correctness on training data also decreases as noise is increased. Therefore, the difference between training correctness and test correctness is much smaller for EITHER-PRUNE (26% for animals at noise-level 0.2) than for EITHER-FULL (66% for animals at noise-level 0.2). This confirms that EITHER-PRUNE avoids much of the over-fitting performed by EITHER-FULL. Ideally, a system's performance on training and test data should be the same in order to completely avoid over-fitting.

Since cover truncation prunes computation as well as theory changes, EITHER-PRUNE is also more efficient than EITHER-FULL. Figure 6.2 shows the training time for both systems as noise is increased. EITHER-PRUNE is clearly faster than EITHER-FULL and the difference increases with noise-level since more pruning is taking place at the higher noise levels.

Finally, cover truncation decreases the number of changes to the theory and therefore decreases the over-all complexity of the revised theory as well. Figure 6.3 shows the number of literals (i.e. total number of consequents and antecedents) in the revised theory as a function of noise-level. EITHER-PRUNE clearly produces simpler theories than EITHER-FULL and again the difference increases with noise-level due to increased pruning. However, the fact that EITHER-PRUNE's theory-complexity still increases with noise-level indicates that it is perhaps not doing enough pruning.

EITHER was also tested on the DNA theory presented in Section 2.6. Figure 6.4 shows accuracy vs. noise curves for this data. The results are averaged over 19 trials of 100 training examples and 100 test examples. EITHER-PRUNE only becomes significantly better than EITHER-FULL at a noise level of 0.1. The differences at lower noise levels are not statistically significant. Both versions of EITHER perform significantly better than ID3. EITHER-PRUNE's performance degrades with noise at about the same rate as ID3's. EITHER-PRUNE's training time is consistently about 6% less than EITHER-FULL's.

Consideration of the above results suggests two possible modes for using the EITHER system. In the *tutorial mode*, when the user is deliberately teaching the system new concepts through the input of carefully selected examples, EITHER should be run without noise processing, so that the concepts may be correctly learned. In the *performance mode*, when EITHER is being run against data of unknown quality, noise processing should be invoked to enhance the likelihood that EITHER will acquire concepts that have not been noise-contaminated.

In summary, the noise algorithm used by EITHER has been shown to:

- improve classification accuracy,
- reduce the complexity of the revised theory, and
- reduce the computation time for revising the theory.

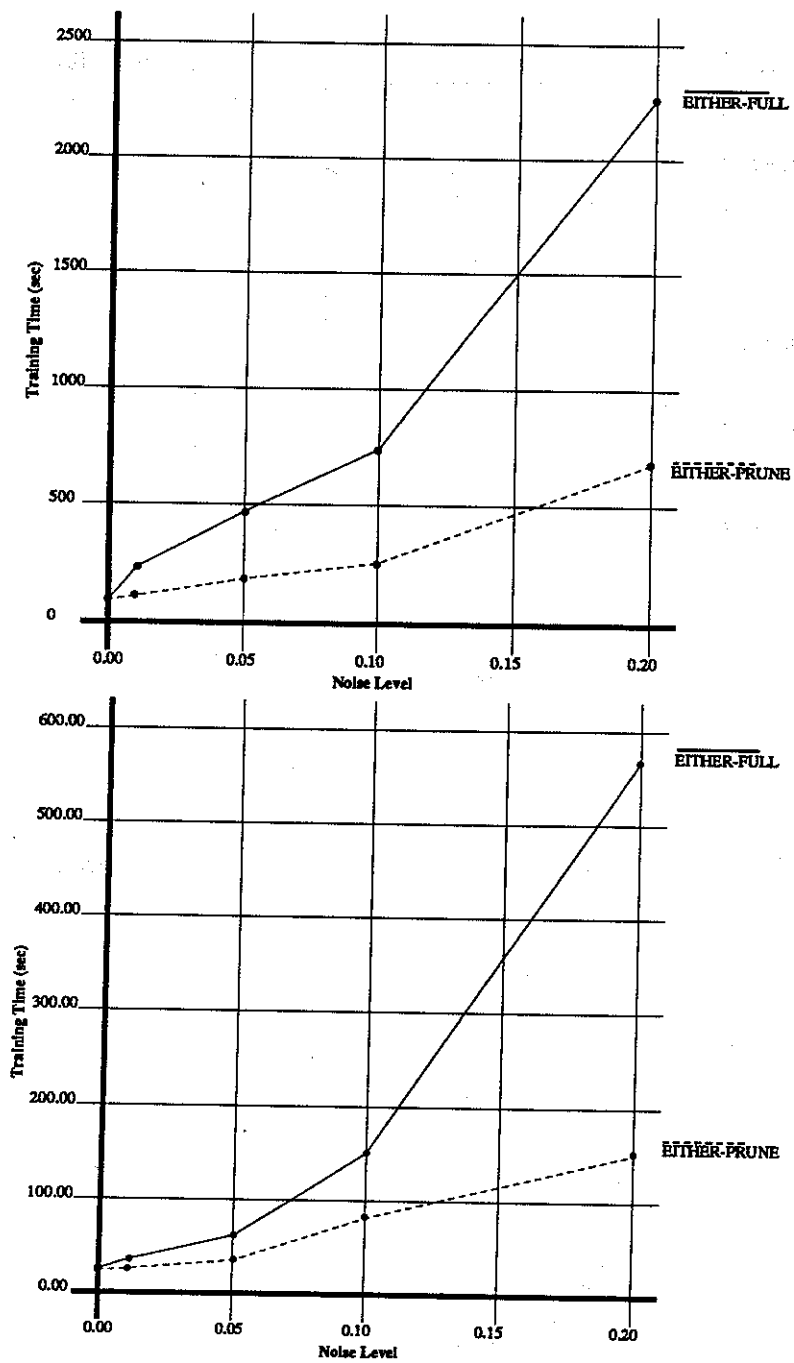


Figure 6.2: Train Time vs. Noise Level for Animal and Computer Domains

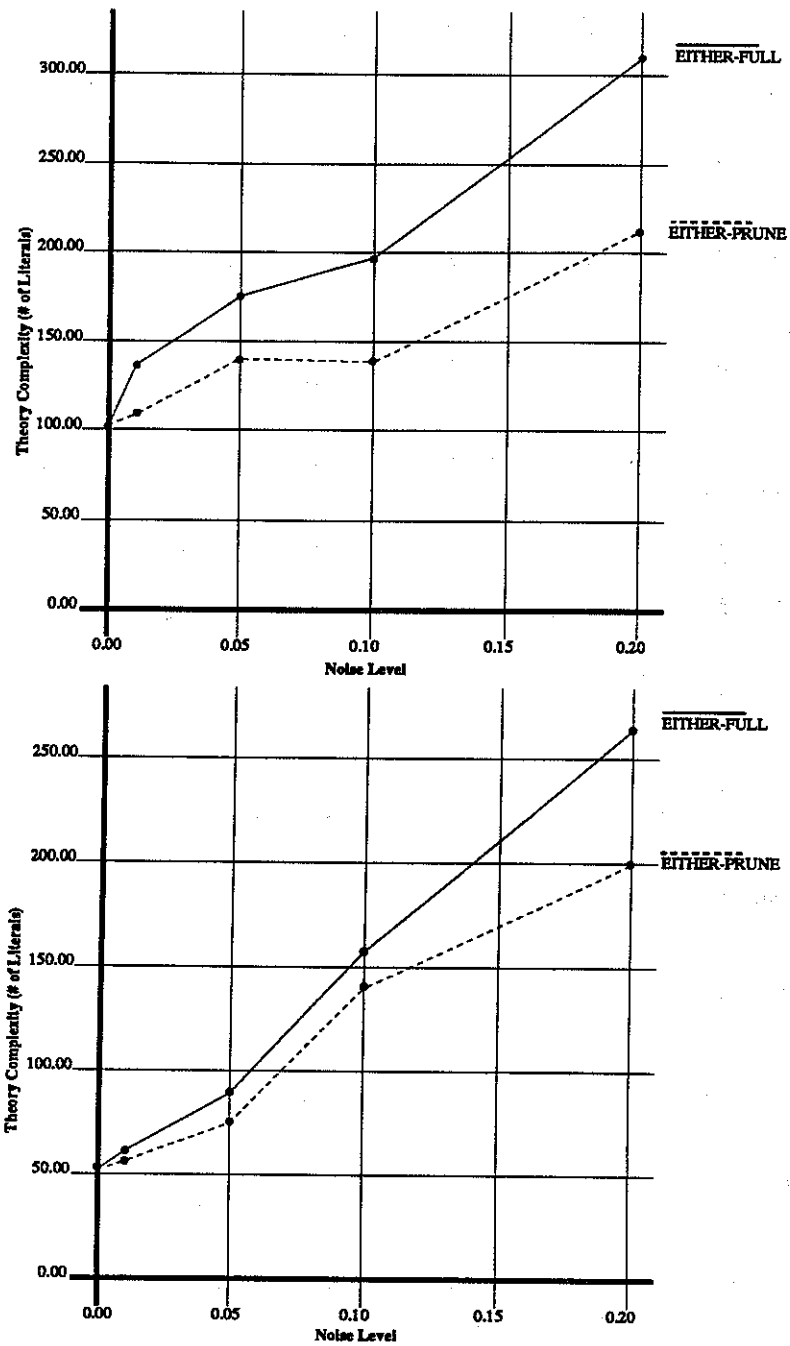


Figure 6.3: Theory Complexity vs. Noise Level for Animal and Computer Domains

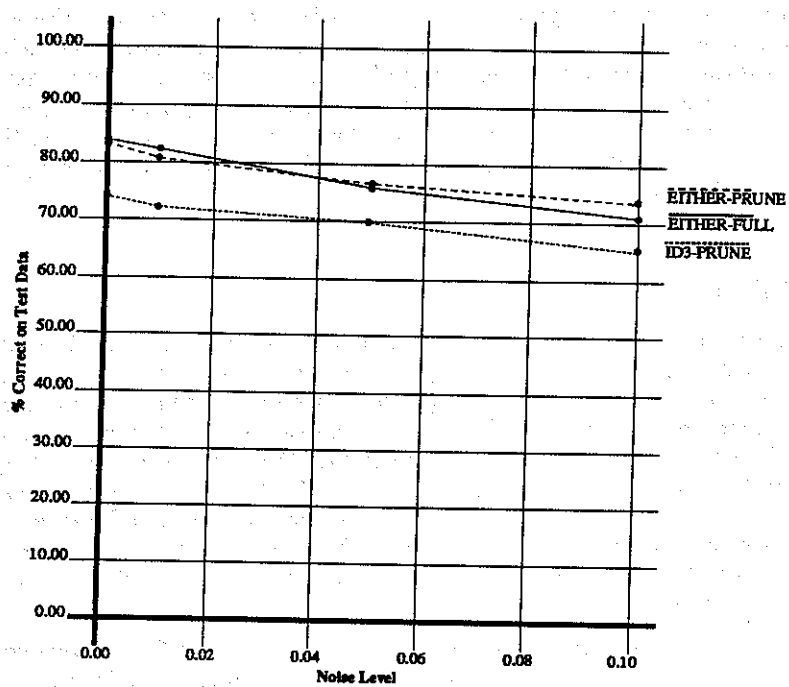


Figure 6.4: Test Accuracy vs. Noise Level for DNA Domain

Chapter 7

Convergence Results

This chapter presents a theoretical analysis of the learnability properties of the EITHER algorithm. The analysis is framed in terms of PAC (Probably Approximately Correct) learnability, a measure that was originally derived by Valiant [Valiant, 1984]. This measure characterizes a learning algorithm in terms of the probability that the error in the hypothesis generated by the learning algorithm will be less than a specified value. Given this measure, conditions under which a learning algorithm will satisfy this measure can be provided. Section 7.1 provides background on the PAC learnability framework as it applies to EITHER, assuming a strictly propositional Horn-clause format for the theory. Section 7.2 shows that the EITHER algorithm will converge to a PAC hypothesis, but may require an exponential number of examples to do so.

7.1 Learnability

The theory of learning proposed by Valiant is an attempt to characterize the inductive bias of a learning algorithm. The approach starts with a definition of a *hypothesis space*, that is the set of all possible hypotheses that can be expressed in the representation language used by the learning algorithm. Within the hypothesis space there is a *version space* that is defined by Mitchell [Mitchell, 1978] as all of the hypotheses in the hypothesis space that are consistent with the training examples. In the case of EITHER, the representation language is propositional Horn-clause logic, and the hypothesis space is the set of all concepts that are representable in Horn-clause logic. A hypothesis within the hypothesis space is consistent with an example when the hypothesis is provable for the example and the example is a member of the hypothesized concept.

The only assumption regarding the examples made by the PAC framework is that both the training and test examples are drawn from the same sample distribution. With this assumption, two measures of learning complexity can be derived within the PAC framework. These are *sample complexity* and *computational complexity*. Sample complexity is derived in this section. Computational complexity is addressed in Chapter 8.

The key result of the PAC learnability analysis is the following (taken from Theorem 4.4 of [Haussler, 1988]):

Let H be a hypothesis space and L be a learning algorithm that uses H consistently (see definition below). For any $0 < \epsilon, \delta < 1$, given

$$m \geq (\ln(1/\delta) + \ln|H|)/\epsilon, \quad (7.1)$$

independent random examples of any target concept c , with probability at least $1 - \delta$, algorithm L will either

- 1. produce a hypothesis in H that has error at most ϵ with respect to c , or*

2. indicate correctly that the target concept is not in H .

An error in a hypothesis is the probability that it classifies the example incorrectly.

In simple terms, the theorem above states that given certain conditions on a learning algorithm, with sufficient samples the learning algorithm will produce a hypothesis that most $(1 - \delta)$ of the time is correct or nearly (within ϵ) correct. The conditions on the learning algorithm are that it uses H consistently, where a learning algorithm uses the hypothesis space consistently if (from [Haussler, 1988], Definition 4.3), for any sequence of examples, Q :

1. if the version space of Q is not empty, then the algorithm produces a hypothesis in the version space,
2. else it indicates that no hypothesis in H is consistent with the given examples.

For EITHER, as mentioned above, the hypothesis space consists of all concepts expressible in Horn clause logic. The version space consists of all concepts expressible in Horn clause logic that are consistent with the input examples. That is, the version space consists of sets of rules which, when augmented with the facts pertaining to a particular example, are capable of proving the category associated with an example, and no other category for all of the examples in the training set.

In the noise-free case, there will always be a hypothesis that is consistent with all of the examples. In the limit, the hypothesis would simply be a set of rules, where each rule would cover a single example, with the rule antecedents being the features of corresponding example, and the rule consequent being the category for the example. However, since EITHER updates an existing theory, we will show that the updates made by EITHER are consistent with the training examples, given this framework. Since we are dealing with the noise-free case in this analysis, it is sufficient to show that the hypothesis returned by EITHER is consistent with the examples, which is done in Section 7.2.

In order to derive the size of the hypothesis space for Horn clause theories, we consider a domain theory that consists of rules that have as antecedents ground literals corresponding to example attributes and as consequents arbitrary hypotheses. If there are n possible example attributes, the maximum size of the hypothesis space for this representation language is 2^{2^n} , assuming binary features. Substituting this expression into equation 7.1 yields:

$$m \geq (\ln(1/\delta) + 2^n \ln 2)/\epsilon \quad (7.2)$$

which establishes that given this number of samples the EITHER algorithm will produce a PAC concept definition.

Clearly, this is a worst case upper bound on the number of samples required by the algorithm. For example, assuming $\epsilon = .1$, $\delta = .1$, and $n = 10$ yields that 7120 samples would be required to achieve this level of learning performance. In general, EITHER requires many fewer training examples than would be predicted by Equation 7.2. The main reason for this is that EITHER is learning subconcepts, which typically require a subset of the total feature set for their expression.

Even though it may require an exponential number of training examples, the fact that EITHER converges to a PAC hypothesis is still an important result. What this means is that given enough additional examples, the hypothesis generated by EITHER is going to improve. No other existing theory revision system guarantees consistency with the

training examples, which means they cannot claim convergence. In practice these systems may converge to local minima or miss the correct concept altogether.

However, more work is required to extend the PAC framework to include the case where the knowledge representation is characterized in terms of an existing theory with a small percentage of erroneous rules. In this case, the analysis probably needs to be parameterized in terms of a metric corresponding to the *distance* between the approximate theory and the correct theory.

7.2 An Argument for Consistency

As discussed in the previous sections, in order to prove the PAC learnability of the EITHER algorithm, it is only necessary to show that the hypotheses generated by the algorithm are consistent with the training examples. This section presents an argument for concluding that the hypotheses generated by EITHER are always consistent with the training examples. This is done in two parts, with separate arguments for the generalization and specialization algorithms. Separating the arguments is justified, since the arguments show that there is no required interaction between the generalization and specialization algorithms.

7.2.1 Consistency of the Generalization Algorithm

The generalization algorithm starts with a cover, that consists of a set of rules and their associated conflicting antecedents. Each of these rules has been associated with one or more positive examples, that are unprovable with the initial theory. The next two sections demonstrate that the updates to the theory by the generalization algorithm 1) correct the failing positive examples for which the generalization algorithm is designed and 2) do not introduce any new failing (provable) negative examples. Taken together, these results show that the output of the generalization algorithm is consistent with the training examples.

Inconsistency problem number 1: failing positive examples. Assume that the correction to the theory from the generalization algorithm results in failing positive examples. Either these examples were part of the original set of failing positive examples that were input to the generalization algorithm, or they are additional examples that were caused to fail by the correction to the theory. If the examples are part of the original set this implies that either the antecedent cover is missing a rule or antecedent required for generalization, or a rule correction is not sufficient to cover one of the examples. If the cover is missing a rule or antecedent, then either the partial proofs set input to the cover algorithm did not contain any partial proofs for a particular example, or the minimum cover algorithm did not incorporate all of the rules in a particular proof. In the first case, the proof set input to the minimum cover algorithm is *defined* to have all of the partial proofs of all of the failing positive examples. (A particular proof may be associated with more than one example when both examples require the same partial proof.) In the second case, the minimum cover algorithm *absorbs* entire proofs in the proof set on each iteration. On each iteration EITHER checks to see if a particular rule and antecedent combination are part of the partially completed minimum cover, and if not EITHER adds the rule and antecedent combination to the cover. These contradictions establish that the minimum cover is complete with respect to the rules requiring generalization.

Next, assume that the generalization to a rule is not sufficient to cover a positive example that used the rule in a partial proof. In order to cover a positive example associated with the rule, the consequent of the rule must be provable when the axioms of the theory are augmented with the facts pertaining to the example (that is, when the partial proof for the

example is turned into a complete proof relative to the given rule). There are two ways a rule can be generalized: antecedent generalization and inductive rule formation. Antecedent generalization uses a one-sided generalization algorithm to form the minimum generalization of the conflicting antecedents for the rule. Note that this generalization is only attempted for leaf-level rules. For a rule not to cover a failing positive example when its antecedents have been generalized, one or more of the generalized antecedents must not unify with the facts of the example. However, this contradicts the definition of minimum generalization: each conflicting antecedent has been generalized such that it no longer conflicts with the corresponding fact in the positive examples for the rule. This contradiction shows that the generalized rule must be sufficient to be provable for the failing positive examples.

In the case of inductive rule formation, the rules that are added to the theory are logically equivalent to the decision trees returned by the ID3 algorithm. When there is no noise in the training examples, the decision trees returned by ID3 are guaranteed to correctly classify all of the examples in the training set [Quinlan, 1986a]. This implies therefore, that one or more of the examples that are failing positive examples for a particular rule are not input to the induction algorithm. However, the minimum cover algorithm for EITHER associates all positive examples that use a particular rule in a partial proof with the rule. This guarantees that all positive examples associated with a rule will be input to the rule induction algorithm. This second contradiction disproves the second assertion above, that a rule in the minimum cover has not been generalized sufficiently to cover an associated failing positive example.

Inconsistency Problem Number 2: Provable Negative Examples. A condition of the EITHER algorithm is that the input to the generalization algorithm does not contain any failing negative examples. As a result, any failing negative examples must have been introduced by the corrections to the theory. With respect to the negative examples, the minimum cover is simply a set of rules that have been selected for correction. The choice of these rules is totally independent of the negative examples. As a result, we only have to show that the *corrections* to these rules do not introduce any failing negative examples.

As in the previous section, the two types of generalization are antecedent generalization and inductive generalization. The last step in antecedent generalization is a check of the negative examples with the modified theory. If any negative examples are provable, the antecedent-generalized result is not returned, and inductive rule formation is returned instead. Hence antecedent generalization cannot introduce any failing negative examples.

In inductive rule formation, the positive examples for the induction are the failing positive examples associated with the rule, and the negative examples are a subset of the input negative examples that become provable by the theory when the original rule is replaced in the theory with a version that has *all* of its antecedents removed. Clearly, the version of the rule with all antecedents removed is the most general version possible. Therefore, any negative examples provable by any version of the rule will be included in the set of negative examples input to the inductive learner. As discussed in the previous section, the rules returned by the inductive learner are guaranteed to be consistent with examples input to the inductive learner. Hence, no new provable negative examples can be introduced by the inductively generalized version of the rule.

7.2.2 Consistency of the Specialization Algorithm

In order to show the consistency of the updates provided by the specialization algorithm, we need only show that the minimum rule cover algorithm provides at least one rule used in each proof of the negative examples, and that the rule updates do not result in

any failing negative examples and are sufficient to prove all of the originally failing positive examples. Each proof of a negative example is a single and-tree from the and-or tree which represents the possible proofs of the negative examples. Consequently, removing a single conjunct, that is, a rule, is sufficient to destroy the corresponding proof, which is the reason for the first condition stated above.

In order to establish the completeness of the minimum cover, we will assume there is a missing term (that is, a rule used in one or more proofs of negative examples) and derive a contradiction. In order for there to be a missing term in the minimum cover, either the proof set input to the minimum cover must not have included a particular proof of one or more of the negative examples, or the minimum cover algorithm must have failed to include any of the rules in a particular input proof. Considering the first case, the proof set input to the minimum cover algorithm is *defined* to be all of the proofs of all of the negative examples. (More than one example may correspond to a single proof, if both examples used the same proof.) Hence the first condition is in contradiction with this definition. In the second case, at each iteration, the minimum cover algorithm selects a rule that appears in one of the proofs in the proof set. The proof set then has this proof removed and any other proof that relied on the same rule. As a result the minimum cover is guaranteed to have at least one rule from every proof in the proof set, in contradiction with the second condition.

The next two sections show that the corrections made by the specialization algorithm 1) remove the proofs of the failing negative examples and 2) still permit the positive examples to be provable by the updated theory.

Inconsistency Problem Number 1: Failing Negative Examples. In order to establish that no failing negative example is provable by the updated theory, we need only show that the updates to the cover rules preclude their being provable for any of the negative examples associated with the cover rules, and that no new negative examples are introduced by the updates. The second condition can be disposed of immediately. All specialization updates to the cover rules are in the form of *strict* specializations: either the rule is removed from the cover (that is the maximum specialization), or existing antecedents are specialized (one-sided antecedent specialization), or new antecedents are added to the rule (inductive rule specialization). In no case are antecedents removed from the rule or traded for other literals with different predicates.

There are two methods of specializing rules in the minimum rule cover: *antecedent specialization*, and *rule retraction*. Clearly, rule retraction will eliminate any proofs that depended on the rule. Hence rule retraction is guaranteed to eliminate the proofs of any negative examples associated with the rule.

Antecedent specialization can be done in two ways, *one-sided antecedent specialization* and *inductive rule specialization*. One-sided antecedent specialization finds an antecedent (if possible) that can be specialized such that the values specified by the antecedent no longer match the feature values associated with the failing negative examples for the rule. Assume that there is a negative example that has a proof that depends on the rule as modified by antecedent specialization. Then all of the antecedents for the rule must unify with the facts pertaining to the example. However, this contradicts the definition of antecedent specialization just given. Hence the antecedent-specialized rule cannot participate in a proof of a negative example. If antecedent specialization is not possible, or if the one-sided specialization causes positive examples to fail with the updated theory, inductive rule specialization is invoked.

Assume that inductive rule specialization has been performed and that a negative example still has a rule proof involving the updated version of the rule. Since the inductively returned rules are consistent with the examples that are used in the induction, this implies

that the same example is both a positive and negative example for the rule, which is not possible unless noise is present in the input. Since we have previously assumed that no noise is present in the input, this implies that inductive rule specialization cannot result in a rule proof for a negative example that involves the inductively specialized form of the rule.

Since no rule proof for a negative example is possible using rules that have been specialized using either retraction, one-sided specialization, or inductive specialization, no proof of the failing negative examples is possible with the corrected theory.

Inconsistency Problem Number 2: Unprovable Positive Examples. There are three parts of the specialization algorithm to consider in regard to failing positive examples: the minimum rule cover algorithm, the one-sided antecedent specialization algorithm, and the inductive specialization algorithm. The rules that are provided by the minimum rule cover are those rules that were used in the proofs of the failing negative examples. Since these rules are arbitrary with respect to the positive examples, the specialization algorithm must guarantee that the correction with respect to each rule does not result in any failing positive examples. Since there are no failing positive examples in the input to the specialization algorithm (that is, the theory has been generalized, if necessary, prior to invoking the specialization algorithm), if this guarantee is successful, there will be no failing positive examples in the output from the specialization algorithm.

Considering the two types of rule specialization, the one-sided specialization algorithm can be discarded from consideration immediately, since the last step in this algorithm is to check the updated theory against the positive examples, and if any positive examples fail to be provable, the inductive specialization algorithm is invoked. When inductive specialization is invoked, the positive examples for the induction are determined by removing the overly general rule from the theory, and using the set of positive examples that fail to be proven with the rule removed. As with the generalization algorithm, any specialization to the rule can only cause positive examples from within this set to be unprovable (that is, removing the rule from the theory is clearly the maximum specialization for the rule). Since the inductive algorithm provides a rule that guarantees consistency with the input examples, clearly no positive example will fail to be provable as a result of the inductive specialization.

Chapter 8

Computational Complexity

An important factor associated with any processing algorithm is its *computational complexity*, that is, a measure of the amount of time required to produce results. Parts of the EITHER processing, such as theorem proving and abduction, are known to be of high computational complexity. The purpose of this chapter is to characterize the computational complexity associated with the various parts of the EITHER algorithm. We also present some suggestions for improving the computational complexity of the EITHER algorithm, and end by showing computation time measurements for EITHER in two experimental domains.

8.1 Analysis

In the case where generalization and specialization are required (that is, the worst case), the computational flow for EITHER can be divided into eight segments. These are: abductively computing the partial proofs for the failing positive examples, selecting the minimum antecedent cover, generalizing the rules in the cover, compressing the generalized rules, computing the possible proofs of the failing negative examples, selecting the minimum rule cover, specializing the rules in the cover, and compressing the specialized rules. These segments are performed serially, although the results from one segment can affect the computations in other segments. Each of these segments will be examined in turn to evaluate the computational complexity of the overall algorithm. For the analysis, we will assume that the greedy algorithm is used in the minimum cover computations.

Abductively creating the partial proofs of the failing positive examples is exponential in the size of the theory, even in the propositional case [Selman and Levesque, 1990]. Since the partial proofs are created independently for each failing positive example, the resultant computations will be linear in the number of failing positive examples. The order of the computation is therefore $O(nb^s)$, where n is the number of failing positive examples and s is the size of the theory and b is the branching factor for the theory. The output of abductive portion of the EITHER algorithm is a set of partial proofs for the failing examples. Note that since these partial proofs consider all possible proof paths through the input theory, the number of partial proofs will be exponential in the size of the input theory.

The minimum antecedent cover segment takes the partial proofs from the abduction segment and computes a set of candidate rules for generalization. These rules are selected such that if all of the conflicting antecedents associated with the rules were retracted from the theory, all of the positive examples would be provable. The lower bound on the size of minimum cover is proportional to the size of the shortest length partial proof in the input partial proofs, since if *all* of the antecedents associated with this proof were assumed, clearly all of the positive examples would be provable (in fact all of the examples would be provable). The maximum size of the antecedent cover is bounded by the size of the input theory. The cover computed by the greedy algorithm is guaranteed to be within a logarithmic factor of the minimum cover. Since the greedy algorithm selects (at least) one rule in the output cover

at each iteration, the number of iterations are proportional to the length of the output cover. However, within each iteration, the algorithm must consider all of the remaining proofs in the partial proof set. Consequently, the computational complexity of the greedy algorithm is $O(ps \log s)$, where s is the size of the input theory and p is the number of partial proofs in the partial proof set received from the abductive segment (proportional to b^s). Therefore, the order of the overall processing requirement is $O(sb^s \log s)$.

For each rule in the minimum antecedent cover, the worst case generalization processing occurs when a one-sided generalization is attempted which fails, and inductive generalization is required. The one-sided generalization is linear in the number of positive examples, since it considers each example in turn to create the generalization. Determining if the generalization failed is done by considering all negative examples and determining if any fail with the candidate generalization. This is done by adding the candidate generalization to the input theory and using theorem proving to determine if a negative example is provable with this modified theory. Theorem proving in the propositional case can be shown to be linear in the size of the input theory [Selman and Levesque, 1990]. When this step fails, the failing positive and negative examples are set to ID3 for inductive correction. This correction has been shown to be linear in the size of the input examples. As a result, the computations performed in the generalization segment are log-linear in the size of the input theory and linear in the size of the input examples.

The set of generalized rules are then input to the rule compression algorithm. This algorithm compares each input rule with the other rules in the set for the purposes of absorption, inter-construction and intra-construction. Performing these computations can be shown to be done in time linear in the number of input rules, since a greedy algorithm is used to create the reduced rules.

The next step is computing the possible proofs for the negative examples. Computing all possible proofs can be shown to be exponential in the size of the input theory [Selman and Levesque, 1990]. Hence this computation will be linear in the number of failing negative examples and exponential in the size of the input theory. The output of this step is the set of unique possible proofs, which is exponential in the size of the input theory.

The next step is to calculate the minimum rule cover. Using the same arguments as were used above for the minimum antecedent cover, the complexity of this computation can be shown to be $O(sb^s \log s)$.

For each rule in the minimum rule cover, the worst case computation occurs when one-sided specialization is attempted and fails, and inductive specialization is required. The complexity of the computation in this case is strictly analogous to that analyzed above for generalization.

The set of specialized rules are then input to the rule compression algorithm, which again operates in time linear with the number of input rules.

Table 8.1 summarizes the results of the complexity analysis. In the table, n refers to the number of input examples, s refers to the size of the input theory, and b refers to the branching factor for the theory. Clearly, the bottlenecks in the processing occur in the calculation of the partial proofs and possible proofs. In the case of the partial proofs, heuristic methods have been developed for improving the efficiency of the calculations [Ng and Mooney, 1991] by only providing the k best partial proofs for each example, where k is a parameter which restricts the beam search during the abduction algorithm. Although these techniques can not guarantee to reduce the complexity of the partial proof calculations in the worst case, they have been shown to be effective on many problems of interest [Ng and Mooney, 1991]. In addition, reducing the number of partial proofs would directly impact the minimum antecedent cover calculations as well, by reducing the number of proofs which would have to be considered.

partial proofs	antecedent cover	rule generalization	rule compression
$\mathcal{O}(sb^s)$	$\mathcal{O}(sb^s \log s)$	$\mathcal{O}(ns \log s)$	$\mathcal{O}(s \log s)$

possible proofs	rule cover	rule specialization	rule compression
$\mathcal{O}(sb^s)$	$\mathcal{O}(sb^s \log s)$	$\mathcal{O}(ns \log s)$	$\mathcal{O}(s \log s)$

Table 8.1: Complexity Results

Computing all possible proofs remains an exponential problem. However, it has not proven to be a significant bottleneck in actual EITHER operation. This is probably because EITHER is given an *approximate* domain theory, that is, a theory that is reasonably close to the required theory. Because the theory is nearly correct, in most cases there will not be many proofs of negative examples. Not only does this effect reduce the processing associated with the possible proofs, it also reduces any processing downstream of this calculation, notably the minimum rule cover calculation. The second effect of a nearly correct theory is that there usually are not many failing negative examples, when compared to the total number of input examples.

These considerations indicate that converting the abduction algorithm to a method which provides a reduced set of partial proofs and uses ATMS would be a useful future update for EITHER. It should also be noted in passing that while theorem proving in the propositional case can be shown to be linear in the size of the input theory, the actual algorithm that is implemented in EITHER does not use the more efficient methods, also making this algorithm a prime target for improvement in the future.

8.2 Experimental Results

Figure 8.1 shows the computer time required for EITHER processing as a function of the number of training examples, for the DNA and soybean theories introduced in Sections 2.6 and 4.5, respectively. The results for the DNA theory show an approximately linear relationship with the number of training examples, as was predicted by the complexity analysis. However, the results for the soybean theory appear to be non-linear with respect to the number of training examples. The difference between the curve and the predicted result in this case is probably because the predicted results are worst-case predictions. The test results are based on a limited number of training examples such that the results are probably dominated by overhead processing and hence do not exhibit the asymptotic behavior predicted by the complexity analysis. The results also show the relatively large magnitude of the time required, for example 5872 seconds of Explorer II CPU time for the theory correction corresponding to one hundred training examples in the case of the soybean theory. To put the computation times into perspective, note that the soybean theory has seventy three rules and 325 symbols, and that the DNA theory has eleven rules and seventy six symbols.

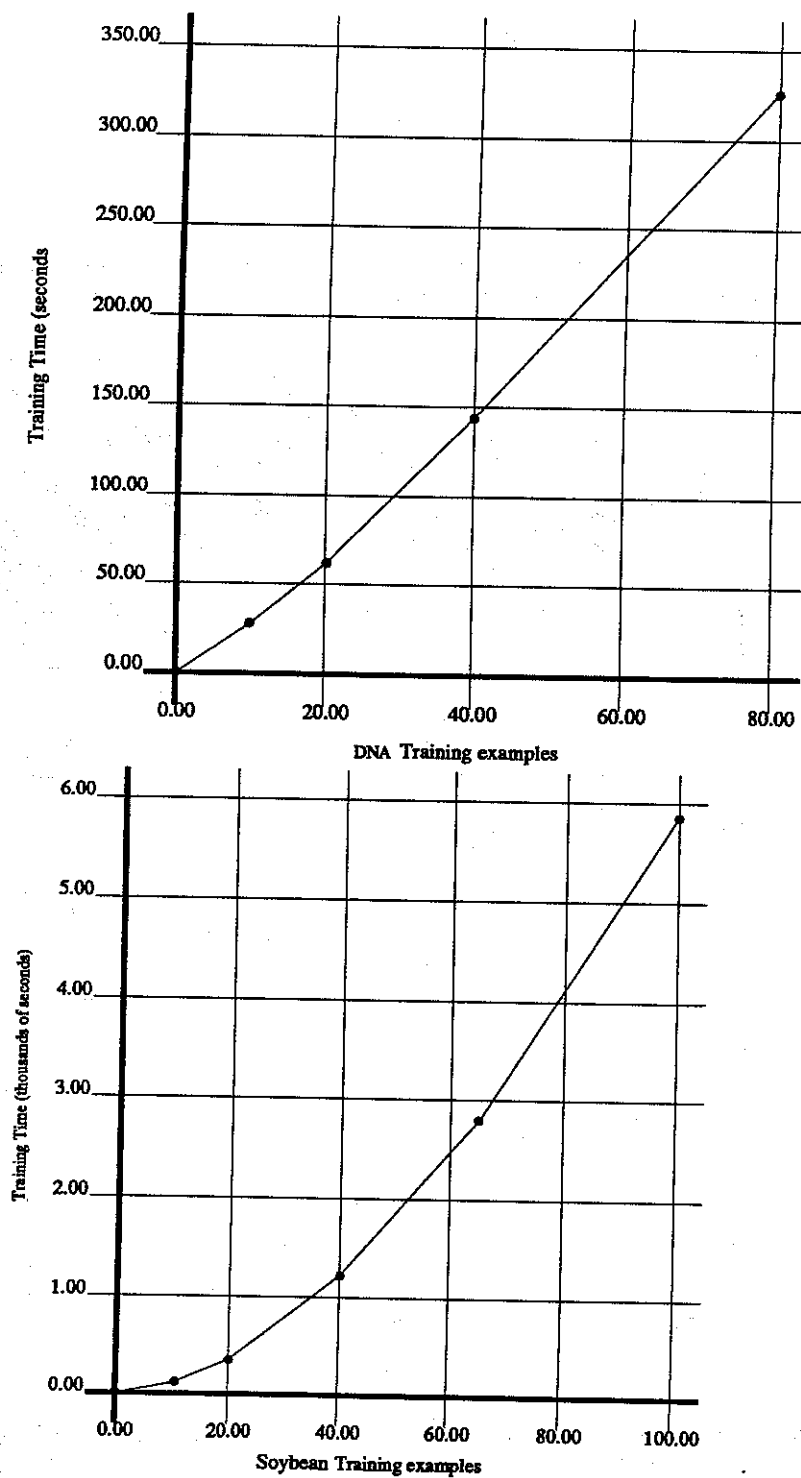


Figure 8.1: Run Time versus Training Set Size for the DNA and Soybean Theories

Chapter 9

Related Work

This section discusses previous work that has been done in the area of theory revision. Figure 9.1 presents a comparison of other work that has been done in the theory revision area.

System	Feature	Overly General Theories	Overly Specific Theories	Modify Domain Theory	Corrections From Multiple Examples	Corrections at Intermediate Points	Disjunctive Rules	Multiple Faults	Predicate Logic
RTLS		X	X	X		X	X		
ML-SMART		X	X		X		X	X	X
FOCL		X	X		X		X	X	X
KBANN		X	X	X	X			X	
IOE		X			X				X
ANA-EBL		X			X				X
IOU		X			X				
LISE			X	X					X
CIGOL			X	X			X	X	X
GEMINI			X	X	X				
ODYSSEUS		X	X	X	X		X		
EITHER		X	X	X	X	X	X	X	

Figure 9.1: Related Work

In general, existing techniques do not address all of the problems of: correcting multiple faults in the theory; changing intermediate level rules; learning disjunctive rules; and handling theories with either overly general or overly specific aspects, or both. In some cases, the techniques do not actually modify the original theory but use alternate data structures

and modify the *results* of the original theory. The techniques which do directly modify the original theory do not guarantee consistency with the training data. Other systems also do not display the modularity displayed by EITHER where, for example, new or improved versions of the induction module can be substituted without affecting the remainder of the system. The remainder of this chapter provides a detailed comparison of EITHER with these other systems.

The criteria for comparing other systems are:

1. Can the system specialize an overly general theory?
2. Can the system generalize an overly specific theory?
3. Does the system modify the input domain theory directly?
4. Does the system use multiple examples to focus the correction?
5. Can intermediate concepts within the theory be learned or corrected?
6. Can the system learn multiple rules for a given concept (implying the use of negative examples in the concept formation)?
7. Can the theory be corrected at multiple points, or is a single fault assumption made?
8. Does the system use full predicate logic in its knowledge representation?

9.1 RTLS

Ginsberg has developed the Reduced Theory Learning System (RTLS) for concept refinement based on a fully expanded theory in minimal sum-of-products (DNF) form [Ginsberg, 1988]. In Ginsberg's approach, the original theory is reduced to a set of cases representing all possible and-trees for the theory expressed in terms of observables. Each hypothesis (goal or consequent) in the theory has a *label* that consists of the sum-of-products expression for it. The reduced theory is then modified so as to make it consistent with the input examples. When the RTLS program was tested against a large Rheumatology data base, it showed improvements in performance of between 17 and 27 per cent.

Since Ginsberg expands the theory prior to correcting it, he does not modify intermediate concepts. This has the effect that all corrections to hypotheses in the theory are done independently. For example, suppose a higher level rule utilizes lower level rules in proofs. If corrections were made to the lower level rules, the same changes would have to be discovered and made to the higher level rule, since all changes are made independently to each concept in the theory.

The RTLS approach is very computationally expensive, since all operations are with respect to the expanded theory, which in the worst case is exponential in the number of rules in the original theory. Ginsberg has developed an algorithm for translating the corrected DNF formula into a multi-layer theory. However, this is done using a top down matching approach to try and determine which rules in the original theory are most likely to match with the corrected form of the expanded theory. Since Ginsberg does not retain the structure of the original theory, this matching process is essentially inductive and may not preserve the original theory structure.

9.2 ML-SMART

The ML-SMART system, [Bergadano and Giordana, 1988], is capable of refining concept descriptions in the presence of incorrect or incomplete theories. ML-SMART forms an *operational* concept description using operational predicates from the domain theory. The process interleaves the expansion of the theory into operational predicates with the testing of the partially expanded theory against applicable examples. In the event that the partially expanded theory fails because some predicate can not be operationalized, the offending predicate is removed from the operationalization process. If the partial expansion covers negative as well as positive examples, then induction is used to continue the theory specialization.

If a partial operationalization does not cover all of the positive examples ML-SMART considers those operationalizations that accounted for the largest increase in the examples covered. It creates new expansions with the predicates involved replaced by the operationalizations and proceeds.

The advantage of ML-SMART is that it interleaves the process of constructing the explanation with correcting a faulty explanation. In addition, it creates explanations that are inductively derived from a set of examples, rather than a single example, as is the case with most other EBL systems. Unfortunately, the corrections that it makes are not mapped back to the underlying domain theory but are only present in the operationalized concept descriptions.

9.3 FOCL

FOCL (First-Order Combined Learner) is a hybrid system that uses Quinlan's FOIL algorithm [Quinlan, 1990] as its inductive component [Pazzani and Kibler, 1990]. FOCL is capable of handling both incomplete and incorrect domain theories. In addition, FOCL is capable of representing relations in the domain theory. The main contribution of the FOCL research is the use of the same information-based metric in the explanation component as is used in the empirical (FOIL) component.

FOCL learns new concepts by *operationalization*, a technique that is similar to that used in ML-SMART. In the operationalization process, antecedents of rules that conflict with examples are expanded using the domain theory until they are expressed in terms of observables. The expanded form of the antecedent, with suitable corrections made, is used to augment the theory.

The primary limitations of the FOCL approach are due to the fact that it relies on operationalizing predicates rather than attempting to find the minimal structural correction to the theory. As a result, FOCL is unable to find intermediate concepts and alters the theory in a way that does not preserve the original structure.

9.4 KBANN

KBANN (Knowledge-based Artificial Neural Networks) [Towell *et al.*, 1990] is an approach to theory refinement that uses the backpropagation algorithm for multi-layer neural networks [Rumelhart *et al.*, 1986] as a method for correcting the domain theory. The technique first translates the existing domain theory into an equivalent neural network, then refines the weights in the network to fit the training examples. It then re-translates the corrected network into an approximately equivalent system of rules.

The KBANN technique can not deal with arbitrary gaps in the theory (where there are no rules for a proving a category or intermediate concept), nor can it introduce new intermediate concepts. These problems could possibly be addressed in KBANN by adding extra hidden units and connections to the initial network; however, this would require predetermining the number and type of intermediate concepts to be created. In addition, KBANN does not guarantee that the revised theory will be consistent with the training examples due to convergence problems associated with the backpropagation algorithm.

9.5 IOE

IOE (Induction Over Explanations) [Flann and Dietterich, 1989] provides a method for identifying a specialized concept within an input domain theory, given a set of positive examples for the concept. IOE constructs the most specific explanation that covers all of the positive examples by finding a minimally generalized explanation that accounts for all of the examples.

IOE does not account for negative examples in this process. However, since the end result is a *specialization* of the original theory, no new negative examples will be provable. IOE also does not attempt to discover intermediate concepts or to invent new higher level rules. In addition, IOE does not account for overly specific theories.

9.6 ANA-EBL

ANA-EBL (ANalogical Abductive Explanation Based Learning) [Cohen, 1990] is a technique for specializing a theory that produces multiple inconsistent explanations for a single example. ANA-EBL finds the smallest set of explanations that cover a set of positive examples while failing to cover a set of negative examples. These explanations are generalized using standard explanation-based learning techniques with the following exception: for each generalized explanation, up to k internal nodes are marked as operational. The use of internal operational nodes accounts for the analogical aspect of ANA-EBL: a test example does not have to match an explanation exactly, but needs only to come within k nodes or less of matching.

ANA-EBL is only capable of correcting overly general theories. In addition, no new intermediate concepts are learned, and the only account that is taken of negative examples is to discard any generalized explanation that covers negative examples. The generalized explanations are used to replace the original rules in the domain theory.

9.7 IOU

IOU (Induction Over the Unexplained) [Mooney and Ourston, 1989] was developed to specialize a theory having overly general aspects to account for concepts representing regularities among examples that are unexplainable using the domain theory. Examples would be manes on lions or children wearing hats at birthday parties, etc. IOU inductively learns the correct subconcepts by processing positive and negative examples of the sub-concept that have been proven by the original domain theory. Only those features of the examples that were not used in the explanations are used in the inductive generalizations.

IOU is only capable of theory specialization. In addition, the form of the specialization is restricted to, in effect, adding antecedents to the top level rule for a category. It is not capable of removing rules as a form of theory specialization.

9.8 LISE

LISE (Learning in Software Engineering) [Genest and Matwin, 1990] provides an approach to theory generalization that uses abduction, analogical reasoning, and case-based reasoning to correct the theory. LISE attempts to complete an explanation using abduction. If this fails, LISE applies analogical reasoning to the problem. Finally, LISE applies case-based reasoning as a last resort.

If abduction is successful, the domain theory is not extended. However, if analogical reasoning is utilized, the domain theory is extended by extracting new rules from the plausible explanation that was used in the analogy. If case based reasoning is required, the program retrieves the case which is most similar to the training example and adapts it to the training example.

LISE only deals with overly specific theories and it only extends them in the case where analogical reasoning is required. It generalizes from a single example and uses only positive examples as input. All corrections are performed at the leaf level of the theory.

9.9 CIGOL

CIGOL (logic spelled backwards) [Muggleton and Buntine, 1988] is a system which was derived from the DUCE [Muggleton, 1987] system for constructive induction. CIGOL provides a series of operators which are derived by taking as input certain combinations of two out of the three elements involved in resolution (the two clauses being resolved upon and the resolvent) and creating the third element. In this way, CIGOL is actually able to invent new predicates which are generalizations of the input predicates. Certain of the operators used by CIGOL have been discussed in Section 3.2.2.

CIGOL assumes an overly specific domain theory and extends it by adding new rules for existing and new concepts. It requires human intervention in the naming and selection of the new predicates. It functions incrementally on single training examples. One advantage of the CIGOL system is that it refines first-order Horn clause theories.

9.10 Gemini

Danyluk is constructing the Gemini system at Columbia. It is a system which learns missing rules in order to fill in gaps in an existing theory [Danyluk, 1989b]. Gemini, like EITHER, uses examples which fail to be proven as indications of missing rules. It organizes the examples in a Generalization Based Memory (GBM) [Lebowitz, 1986a] according to the goal concept, the example, and the partial explanation. In this way, examples are only declared to be similar if they fail using the same partial explanation. Examples which are similar are declared to be part of the *induction set* and are used to learn the new rule. The rule which is learned contains as a consequent the unproven subgoal at the end of the deduction chain (that is, the consequent of the last rule in the chain which cannot be proven). The rule antecedents are the common feature values for the examples in the induction set after the feature set has been reduced using several heuristics.

In Gemini, it is assumed that the initial theory is overly specific. All modifications to the theory are for the purpose of generalization, that is, new rules are added to the theory. It is assumed that all the existing rules are correct. Gemini makes a "single fault" assumption: only one subgoal in the theory is assumed to fail. In addition, Gemini only considers positive examples that fail to be proven.

The major contrast between the Gemini algorithm and the EITHER algorithm is that Gemini learns a single rule whose antecedent is the minimally specific generalization [Haussler, 1988] of the positive examples, and whose consequent is the failed subgoal, whereas EITHER can learn multiple revisions to the existing theory as well as disjunctive rules (since it uses positive and negative examples). In addition, EITHER is capable of theory generalization or specialization.

9.11 OCCAM

OCCAM [Pazzani, 1988a] is capable of correcting incorrect and incomplete theories. OCCAM includes two forms of learning, EBL and SBL. EBL is applied first and empirical methods are only used as a last resort. OCCAM keeps track of the predictions made by rules in the domain theory, and it deletes rules after they have made too many incorrect predictions. Rules are added inductively to the theory as necessary to complete explanations.

OCCAM represents its compiled (EBL) knowledge in the form of schemas. Although the domain theory may have been revised, the schemata are not revised until they make an incorrect prediction. When a schema makes an incorrect prediction, it is deleted and a new schema is acquired using EBL with the correct domain theory.

In acquiring new rules for incomplete explanations, OCCAM uses abstract explanations to focus the induction on the single inference rule needed to complete the explanation chain. Minimal specific generalization is used to find the antecedents of the rule.

OCCAM is also capable of handling the case where a specific rule in the underlying theory is overly specialized and needs to be generalized [Pazzani, 1989]. OCCAM retrieves a schema which would successfully account for the input example, with the exception that some of its input conditions are not met. OCCAM uses this schema as an index into the domain theory to find the inference rule corresponding to the features which failed. Antecedents which do not match the features in the input example are then removed from the premises of the rule (that is, it is generalized in this way).

Limitations of the OCCAM approach are that it only uses a single example for its rule generalizations, it assumes a single point of failure, it cannot learn intermediate concepts, and, since it does not use negative examples, it can only learn a single conjunctive generalization as a new rule.

9.12 ODYSSEUS

ODYSSEUS is a knowledge base refinement system designed to work in conjunction with the HERACLES expert system shell [Wilkins, 1988]. ODYSSEUS constructs explanation chains using meta-rules which act as templates for information which must be filled in during the explanation process. In the event of an explanation failure, the meta-rules serve as a guide in the acquisition of new knowledge for the system. When new knowledge is proposed for the system (in the form of tuples which may represent either rules or frame knowledge), ODYSSEUS uses a confirmation theory to validate the proposed update. ODYSSEUS implements *apprenticeship learning* - it is assumed that a user is involved in critiquing and guiding the proposed updates.

In its corrections, ODYSSEUS always makes a single fault assumption. While multiple examples are used to form and confirm the inductive correction to the theory, a single example is used to identify the location of the error. ODYSSEUS also distinguishes between domain knowledge and strategy knowledge and is only capable of learning domain

knowledge. The refinements which ODYSSEUS makes do not guarantee correct performance over the training examples.

9.13 COAST

COAST (Correcting and Augmenting Scientific Theories) [Rajamoney and DeJong, 1988] is a system which uses experimentation to correct qualitative process theories [Forbus, 1984]. COAST detects problems with the existing theory when it either fails to predict the result of a process or is unable to explain an observation. COAST formulates candidate hypotheses which explain the failure based on the type of failure and constraints from the scenario in which the failure occurred. COAST then devises experiments which will discriminate among the candidate hypotheses. COAST requires that any proposed theory refinement be consistent with exemplars representing prototypical cases for the correct theory. The best theory among the proposed candidates is selected based on syntactic simplicity, simplicity of explanations, and predictive power. COAST has successfully revised theories of fluid flow, combustion and evaporation.

COAST uses experimentation, rather than a set of positive and negative examples to determine the proposed extensions to its theory. It then uses exemplars to *reject* inconsistent candidate theories, rather than as tools in the creation of the revisions. All of its exemplars are positive examples, and hence COAST does not make use of negative examples in its theory revision. COAST is based on qualitative process theories. It does not deal with domain theories expressed in propositional or predicate logic.

9.14 KI

The KI (Knowledge Integration) program [Murray, 1989] is being developed at the University of Texas to aid in the knowledge acquisition task. Its purpose is to incorporate new knowledge consisting of facts or rules into an existing knowledge base. The process of knowledge integration is made up of the three phases: recognition, elaboration, and adaptation.

In the recognition phase, KI attempts to determine which pieces of the existing knowledge base are relevant to the new knowledge. In the elaboration phase the training knowledge is expanded using the existing knowledge base to determine its impact on the knowledge base and to reveal any inconsistencies. The adaptation phase consists of modifying the knowledge base to account for the new knowledge and any inconsistencies that have been revealed.

KI assumes abstract rules as input, rather than specific training examples. Its purpose is to help the user update the knowledge base, rather than to independently learn new concepts. KI makes a single fault assumption and works with a single example (rule) at a time. KI assumes direct interaction and guidance from an expert. It is possible for KI to have inconsistent knowledge in the knowledge base.

Chapter 10

Future Work

Several promising areas for future research have been discovered in the course of the current research. Primarily, identified work has to do with either the knowledge representation used by EITHER, or the efficiency of the EITHER implementation. Each of these areas will be discussed separately.

10.1 Improving the Knowledge Representation

An obvious limitation of the current EITHER knowledge representation is the restriction to propositional Horn clause logic. While converting EITHER to a full first order logic is probably a large project, a first step in this direction would be to extend the knowledge representation to a Horn clause theory formulation involving full predicate logic [Richards and Mooney, 1991]. This would allow reasoning about relations when updating the theory. The abduction and deduction components used by EITHER could easily be updated to handle this requirement. A promising component for the inductive learner in this case would be to incorporate the FOIL algorithm, which is an expansion of the information-theoretic approach used by ID3 to first-order Horn Clauses. Within the EITHER algorithm, the main work would be in the specialization and generalization components, which would have to be updated to handle variables that represent entities.

A second shortcoming in EITHER's current knowledge representation is an inability to reason probabilistically. This limits the domain theories which can be corrected using EITHER, since many existing domain theories are expressed probabilistically. A first step in that direction was the incorporation of a fuzzy tester, described in Section 4.5. However to fully capture the meaning of domain theories that are described probabilistically, the knowledge representation needs to include probabilistic weights for rules. In addition, the formalism for the probabilistic reasoning must be chosen. The general complication that probabilistic reasoning introduces is that now when considering a rule update EITHER must decide whether to update the probability associated with the rule, or the rule itself, or both.

A third problem with the current knowledge representation concerns the lack of an ability to reflect the degree of belief in proposed theory revisions. Currently, all changes made by EITHER are reflected directly in the theory. The effect of EITHER's confidence in the proposed change is reflected in the *type* of change. For example, when EITHER lacks sufficient evidence for a rule retraction, it will instead specialize the rule's antecedents.

Perhaps a better approach is to include abduction as an option in the spectrum of responses to an incorrect theory. For example, if only a single example has been seen by the system, probably no permanent change to the theory is warranted on the basis of the example. However, what is warranted is an *assumption* regarding the current state of the example/theory combination. That is, we may choose some minimal assumption that corrects the conflict between the theory and the example. However, at the time of making the assumption we make no assertion as to the source of the discrepancy - whether the example is

in error or the theory is in error. It is only in the light of future evidence that an assumption is assigned as either a theory or example error. Using this formalism EITHER would replace the current approach, which always corrects discrepancies between examples and theory with a correction to the theory, with a sliding scale in which discrepancies with little evidence are assigned as assumptions with a minimal degree of belief. Only after considerable evidence has been accumulated (that is, an assumption with a high degree of belief that it corresponds to a defect in the theory) is a change actually made to the theory. On the other side, if, after sufficient processing an assumption is still assigned to an example (that is, it still has a minimal degree of belief), then the example should be removed from the training examples.

A final knowledge representation update is due to the representation for examples used by EITHER. The current version of EITHER assumes that all of the categories associated with the examples are disjoint. This causes a limitation on the directed learning of intermediate concepts, since examples of such concepts are actually examples of the given concept *and* all of the superior concepts that are true for the example and depend on the intermediate concept. Hence an improvement to the program capability would be to allow the labeling of examples as members of multiple, possibly intermediate, concepts, rather than just a single category. In fact, it may prove useful to explicitly label examples with the concepts to which they *do not* belong. This would be useful when the example is a near-miss for another concept and should be used first in rule formation. If the corrected rule based on near-misses did not introduce any failing examples, then the correction would be complete. If additional failing examples were introduced, these could be used for a final correction to the rule.

10.2 Efficiency Improvements

A final area for future work is in improving the efficiency of the EITHER algorithm. Although considerable attention was paid to this issue in the current version of EITHER (primarily in the use of greedy cover algorithms for both the minimum rule and minimum antecedent covers), there is still significant room for improvement. A major improvement could be achieved in the abduction algorithm where the current version calculates all partial proofs for all of the examples. A more efficient version would save sub-proofs using an ATMS approach, [Ng and Mooney, 1991], and only compute a new sub-proof where one did not exist. This new version of ABDUCE would also only compute a subset of all of the possible partial proofs, using a beam search technique. Another possible speed improvement could be achieved by parallelizing the rule updates. For example, the one-sided generalization and specialization algorithms do not depend on the updated state of the theory but could be done in parallel based on the input version of the theory. As was shown in Chapter 6, only computing a partial cover can also greatly reduce processing time, while only resulting in a small decrease in performance, and this method needs to be investigated further. The deduction algorithm could be improved, for propositional theories, to require processing time which is linear in the size of the theory rather than exponential, which is currently the case.

Chapter 11

Conclusions

The thesis for the research reported in this dissertation is the following:

Using explanations from an incorrect theory to focus inductive corrections to the theory can result in corrections that are superior both syntactically and in terms of performance.

Here by explanations we mean *partial* explanations, in the case of an overly specific theory, or *possible* explanations, in the case of an overly general theory. What do we mean by "superior?" For both syntactic and performance superiority, the reference point is an inductive learning system. Syntactic superiority is demonstrated by the fact that the corrections are expressed in the same representation language that was used to construct the theory, and that the corrections are minimal with respect to the initial theory. Inductive corrections are expressed in some *operational* language, and do not take advantage of any initial theory. In terms of performance, superiority refers to superiority of classification performance. The use of an initial theory can provide a performance advantage, even in cases where the initial theory is not able to correctly classify a *single* example.

The method reported in this dissertation is also able to make "reasonable" corrections to a theory even when the training examples given to the method are noisy. This is done by using a simple method that rejects corrections to the theory that are necessitated by a single example. The rationale for this approach is that if the correction only corresponds to a single example, it is more likely that the example is in error than the theory.

The algorithm for theory correction presented in this dissertation will eventually converge to a hypothesis that is within a small error of the correct hypothesis. This is an important result in that we can depend on the stability of the algorithm: predictions made by the corrected theory are guaranteed to improve as sufficient additional training examples are given to the algorithm.

The complexity of the current algorithm has been shown to be exponential in the size of the input theory. This is an unfortunate result. The two main causes of this effect are the abductive generation of the partial proofs used by the generalization algorithm, and the generation of all possible proofs of negative examples, where the possible proofs are used by the specialization algorithm. For the abductive algorithm, the computational bottleneck can be relieved by the use of an ATMS approach. There is no relief for the bottleneck associated with the possible proofs, provided that we require complete consistency with the training examples. However, considerable improvement in computational efficiency for the possible proofs can be obtained if we relax the consistency requirements. This was demonstrated in the noise resistance algorithm where providing a partial cover for the negative examples, if done right, provided a large gain in computational efficiency with only a small decrease in performance.

Assertions regarding the algorithm's performance have been verified in a variety of problem domains. These experimental results show that the use of an approximate theory

provides a significant performance advantage when compared to purely inductive learning. In addition, an examination of the types of changes made by the algorithm in these cases show that the revisions correct multiple faults, correct and discover intermediate concepts within the theory, and are capable of correcting both specialization and generalization errors. The corrections that are made tend to preserve the original structure for the theory if at all possible. A benefit of the ability to learn intermediate concepts is the possibility of *inter-category transfer*, where learning in one category actually improves performance in another related category.

Appendix A

The EITHER algorithm

This appendix presents the definition of the EITHER algorithm.

Algorithm either(T, E)

Input:

An approximately correct theory, T , and a set of training examples, E .

Output:

An updated theory, T' , which is consistent with the training examples.

begin

let

$P \leftarrow \text{failed-positives}(E, T)$

$N \leftarrow \text{failed-negatives}(E, T)$

$GT \leftarrow \text{generalize}(T, P, (E - N))$

$N \leftarrow \text{failed-negatives}(E, GT)$

$T' \leftarrow \text{specialize}(GT, N, E)$

return T'

end

Algorithm generalize(T, P, E)

Input:

An overly specific theory, T , a set of failing (unprovable) positive examples, P , and a set of remaining examples, E , none of which are provable negatives with the initial theory.

Output:

A generalized theory, T' , which properly classifies all of the positive examples, and which does not introduce any new failing negative examples.

begin

let

$IC \leftarrow \text{minimum-antecedent-cover}(P, T)$

Initial cover from minimum antecedent cover algorithm.

$CC, CT \leftarrow \text{corrected-cover}(IC, T, P, E)$

Updates to rules which insures that all rules are correctable.

$SC, ST \leftarrow \text{simplest-rules}(CC, CT, P, E)$

Syntactic reduction to rules.

$RR, T' \leftarrow \text{compress-rules}(SC, ST)$

A compressed version of rules using inverse resolution operators.

return T'

end

Algorithm specialize(GT, N, E)

Input:

An overly general theory, G , a set of failing negative examples, N , and a set of remaining examples, E , where there are no failing positive examples in the set E .

Output:

A specialized theory, T , in which there are no failing positive or negative examples.

begin

let

$IC \leftarrow \text{minimum-rule-cover}(N, T)$

Initial cover from minimum rule cover algorithm.

$CC, CT \leftarrow \text{corrected-cover}(IC, T, P, E)$

Updates to rules which insures that all rules are correctable.

$SC, ST \leftarrow \text{simplest-rules}(CC, CT, P, E)$

Syntactic reduction to rules.

$RR, T' \leftarrow \text{compress-rules}(SC, ST)$

A compressed version of rules using inverse resolution operators.

return T'

end

Algorithm corrected-cover(C, T, F, E)

Input:

A minimum cover, C , an incorrect theory, T , a set of failing examples, F , and a set of remaining examples, E .

Output:

A revised cover C' , and associated theory, T' , where the rules in C' have no correctability problems.

begin

let

$C' \leftarrow C$

for $r \in C'$

let

$r' \leftarrow \text{revise-rule}(r, F, E, T)$

if $\neg(\text{overlapping-examples}(E, \text{modify-theory}(r', r, T), F) = \emptyset)$

Check to see if there is overlap between the examples input for the rule correction (F), and the examples which fail after the rule has been corrected.

```

    then
       $C' \leftarrow$ 
      corrected-cover(substitute(parents( $r$ ),  $r$ ,  $C'$ ),  $T$ ,  $F$ ,  $E$ )
       $\cup (C' - \{r\})$ 
      If so, replace  $r$  with its parent rules in the current cover and re-
      curse.
    else
       $C' \leftarrow$  substitute(revise-rule( $r$ ,  $F$ ,  $E$ ,  $T$ ),  $r$ ,  $C'$ )
      Else substitute the corrected version of the rule in  $C'$  and continue.
  end-for
  return  $C'$ 
end

```

Algorithm simplest-rules(C , T , F , E)

begin

let

$C' \leftarrow C$

$C'' \leftarrow$ *<A set of grouped rules in C , each rule group having a single, common parent.>*

for $g \in C''$

if *<The rules in g required inductive correction>*

then

if simpler(revise-rule-group(parents(g), F , E , T),
revise-rule-group(g , F , E , T))

then

substitute(parents(g), g , C')

end-for

if $C' \equiv C$

then

return C

else

simplest-rules(C' , T , F , E)

end

Algorithm compress-rules(C , T)

return

The compressed version of the rules in C , using the inverse resolution operators absorb, intra and inter.

Algorithm revise-rule(r , F , E , T)

Input:

An incorrect rule, r , a set of failing examples, F (either positive or negative), a theory, T , and a set of remaining examples, E .

Output:

A corrected version of the rule which correctly classifies all of the examples in F , and does not incorrectly classify any of the examples in E .

begin

```

if <F consists of failing positive examples.>
then
   $r \leftarrow \text{revise-specific-rule}(r, F, E, T)$ 
else
   $r \leftarrow \text{revise-general-rule}(r, F, E, T)$ 
return r
end

```

```

Algorithm overlapping-examples(E,T,F)
begin
if failed-positives(F)
then
  return failed-negatives(E, T)  $\cap$  F
else
  return failed-positives(E, T)  $\cap$  F
end

```

```

Algorithm modify-theory(r',r,T)
if <r' is a generalized rule>
then
  return  $T \cup \{r'\}$ 
else
  return substitute(r', r, T)

```

```

Algorithm substitute(x,y,r)
return r, with x substituted for y in r.

```

```

Algorithm simpler(x,y)
if <The rule changes in x are syntactically simpler than the rule changes in y>
then
  return True
else
  return False

```

```

Algorithm revise-rule-group(G,F,E,T)
begin
for  $r \in G$ 
   $G \leftarrow \text{substitute}(\text{inductively-correct-rule}(r, F, E, T), r, G)$ 
  Form an inductive correction for each rule in the rule group, G.
end-for
return G
end

```


Algorithm parents(*g*)

return *The parents of the rules contained in g.*

Algorithm revise-specific-rule(*r, P, E, T*)

Input:

An overly specific rule, *r*, a set of failing positive examples, *P*, which fail to be provable with theory *T* using the current version of *r*, and a set of remaining examples, *E*.

Output:

A corrected version of the rule whose premises are satisfied by all of the examples in *F*, and which does not result in any failing negative examples among the examples in *E*, when added to theory.

$r' \leftarrow \text{remove-conflicting-antecedents}(r)$

if $\neg \text{failed-negatives}(E, T \cup \{r'\}) \equiv \emptyset$

then

$r' \leftarrow \text{generalize-antecedents}(r, P)$

if $\neg \text{failed-negatives}(E, T \cup \{r'\}) \equiv \emptyset$

then

$r' \leftarrow \text{inductively-correct-rule}(r, P, E, T)$

return r'

end

Algorithm revise-general-rule(*r, N, E, T*)

Input:

An overly general rule, *r*, a set of failing negative examples, *N*, which are provable with theory, *T* using the current version of *r*, and a set of remaining examples, *E*.

Output:

A corrected version of the rule whose premises are not satisfied by any of the examples in *N*, and which does not result in any failing positive examples among the examples in *E*, when substituted for *r* in the theory.

if $\text{failed-positives}(E, T - \{r'\}) \equiv \emptyset$

then

if *<There are "many" positive examples in E>*

then

$r \leftarrow \text{deleted}$

else

$r \leftarrow \text{one-sided-specialization}(r, N, T)$

else

if $\text{one-sided-specialization}(r, N, T)$

then

$r \leftarrow \text{one-sided-specialization}(r, N, T)$

else

$r \leftarrow \text{inductively-correct-rule}(r, N, E, T)$

return r
end

Algorithm failed-negatives(E, T)

Input:

A set of examples, E , and a theory, T .

return

The set of negative examples in E provable using T , in the case of single category theories, or the set of examples provable in other than their own category using T , in the case of multi-category theories.

Algorithm failed-positives(E, T)

Input:

A set of examples, E , and a theory, T .

return

The set of positive examples in E which are not provable using T , in the case of single category theories, or the set of examples not provable in their own category using T , in the case of multi-category theories.

Algorithm generalize-antecedents(r, P)

Input:

A rule, r , and its associated conflicting antecedents, and a set of positive examples, P .

Output:

The rule, r , with all conflicting antecedents generalized, if possible. If it is not possible to generalize all antecedents, then *false* is returned.

begin

for $a \in \text{conflicting-antecedents}(r)$

Generalize each antecedent in turn.

$a' \leftarrow \text{generalize-antecedent}(a, P)$

if $\neg a'$

then

return *false*

else

$r \leftarrow \text{substitute}(a', a, r)$

end-for

return r

end

Algorithm one-sided-specialization(r, N, T)

Input:

A rule, r , a set of failing negative examples, N , and an input theory, T .

Output:

The rule r with antecedents which do not match any of the examples in N , if possible, or *false*.

let

$a' \leftarrow \text{false}$

```

    a ← false
    a', a ← minimally-specialized-literal(antecedents(r), N, T)
    if ¬a'
    then
        a', a ← minimally-specialized-literal(observables() - ant(r), N, T)
    if a'
    then
        return r ← substitute(a', a, r)
    else
        return false
end

```

Algorithm generalize-antecedent(a,P)

return

Either the antecedent a generalized so as to be provable for all of the positive examples, P, or false.

Algorithm minimally-specialized-literal(L,N,T)

return *A minimal subset of the literals in L, specialized so as not to be provable for any of the examples in N.*

Algorithm inductively-correct-rule(r,F,E,T)

begin

let

$P \leftarrow \emptyset, N \leftarrow \emptyset$

if failed-positives(F)

then

$P \leftarrow F$

$N \leftarrow \text{failed-negatives}(E, T \cup \text{remove-antecedents}(r))$

else

$P \leftarrow \text{failed-positives}(E, T - \{r\})$

$N \leftarrow F$

return inductive-correction(r, forward-chain(P; r, T), forward-chain(N, r, T))

end

Algorithm inductive-correction(r,P,N)

return

A rule, with the same consequent as r, and whose antecedents are formed using an inductive hypothesis which is true for P and not for N.

Algorithm forward-chain(E,r,T)

return

The examples in E , with additional features for each example consisting of the concepts in T which were provable for the example, and excluding any concept transitively reachable in $\text{parents}(\{r\})$.

Algorithm remove-antecedents(r)

return <The rule r with all antecedents removed. >

Appendix B

The Minimum Antecedent Cover Algorithm

The minimum antecedent cover algorithm based on the greedy cover algorithm is as follows:

Algorithm minimum-antecedent-cover(E, T)

Input:

A set of unprovable positive examples, E , and an overly specific theory, T .

Output:

A set of rules, R , and associated conflicting antecedents, such that if all the conflicting antecedents in all the rules in R are retracted from their associated rules, the set of unprovable positive examples will be provable.

begin

 let $P = \text{proofs}(E, T)$

$R = \text{minimum-antecedent-cover1}(E, P, \emptyset)$

 return R

end

Algorithm proofs(E, T)

Input: A set of positive examples, E , that are unprovable in the theory, T .

begin

 return The set of all possible partial proofs of E in the domain theory.

end

Algorithm minimum-antecedent-cover1(E, P, C)

Input:

A set of examples, E , remaining to be covered.

A set of partial proofs, P , for the examples in E .

A partial cover, C , consisting of antecedent retractions.

begin

 if $E = \emptyset$

 then

 return C

 else

 let $btc \leftarrow 0$

Benefit to cost.
 $CP \leftarrow \emptyset$
Cover proofs.
 $CE \leftarrow \emptyset$
Cover examples.
 $BC \leftarrow \emptyset$
Best cover.
for $p \in P$
let
 $P^- \leftarrow P - \{p\}$
Remove selected proof from current proof set.
 $TP \leftarrow \{p\}$
Proof to be tentatively added to current cover in search
 $TC \leftarrow \text{rules}(p) \cup C$
Tentative cover
 $TE \leftarrow \emptyset$
Initialize examples covered by current cover to empty set.
for $p^- \in P^-$
if $\text{antecedents}(p^-) \subseteq TC$
then $TE \leftarrow TE \cup \text{examples}(p^-)$
 $TP \leftarrow TP \cup \{p^-\}$
endfor
if $btc < \text{benefit-to-cost}(TC, TE)$
then
 $btc \leftarrow \text{benefit-to-cost}(TC, TE)$
 $CP \leftarrow TP$
 $CE \leftarrow TE$
 $BC \leftarrow TC$
endifor
minimum-antecedent-cover1($E - CE, P - CP, BC$)
end

Algorithm benefit-to-cost(C, E)

begin

return $|E| / |C|$

end

Algorithm rules(p)

begin

return The rules used in the proof containing antecedents
that must be retracted in order for the proof to
be completed.

end

Algorithm examples(p)

Input: partial proof specified in terms of the domain theory.

begin

return the unprovable positive examples that can
be proven if the conflicting antecedents for

end the partial proof are retracted.

Appendix C

The Minimum Rule Cover Algorithm

This appendix contains two sections which define the greedy and exhaustive versions of the minimum rule cover algorithm.

C.1 The Greedy Rule Cover Algorithm

Algorithm minimum-rule-cover(E)

Input:

A set of sets, E , where each element consists of the rules used in a particular proof of an example

Output:

A minimum cover.

```
begin
   $R \leftarrow \text{maximum-rule-factor}(E)$ 
   $F \leftarrow \text{factored-terms}(E, R)$ 
  if  $RE = \emptyset$ 
    then return  $\{R\}$ 
  else return  $S \leftarrow \{R\} \cup \text{minimum-rule-cover}(RE)$ 
end
```

Algorithm maximum-rule-factor(E)

```
begin
  let  $R$  be the rule found in the greatest number of elements of  $E$ 
   $N$  be the number of elements of  $E$  containing  $R$ 
  if  $N = 1$ 
    then
      return false
    else
      return  $R$ 
end
```

Algorithm factored-terms(E, R)

```
begin
  let
     $E'$  be the set of all elements of  $E$  that contain  $R$ 
     $O \leftarrow \emptyset$ 
```



```

    fore  $\in E'$ 
       $O \leftarrow (e - \{R\}) \cup O$ 
    endfor
  return  $O$ 
end

```

C.2 Exhaustive Rule Cover Algorithm

Algorithm minimum-rule-cover(E)

Input:

A set of sets, E , where each element consists of the rules used in a particular proof of an example

Output:

A minimum cover.

return $\text{lef}(\text{exhaustive-cover}(E))$

Algorithm $\text{lef}(C)$

Input:

A set of covers, C .

return

< The cover with the minimum value for the lef metric ordered by the criteria of: number of overlapping examples, number of unprovable positive examples, and cover length. >

Algorithm exhaustive-cover(E)

Input:

A set of sets, E , where each element consists of the rules used in a particular proof of an example

Output:

A minimal set of rule covers, S , each cover consisting of a set of rules, such that if all of the rules in the cover are retracted, all currently provable negative examples are guaranteed not to be provable.

begin

$R \leftarrow \text{maximum-rule-factor}(E)$

$F \leftarrow \text{factored-terms}(E, R)$

$RE \leftarrow E - F$

if $\neg R$

then $S \leftarrow \text{cartesian-product}(E)$

else

if $F = \emptyset$

then $S \leftarrow \text{scalar-product}(R, \text{exhaustive-cover}(RE))$

else

if $RE = \emptyset$

```

        then  $S \leftarrow \{R\} \cup \text{exhaustive-cover}(RE)$ 
        else  $S \leftarrow (\{R\} \cup \text{exhaustive-cover}(RE))$ 
            $\cup \text{exhaustive-cover}(F \cup RE)$ 
      endif
    endif
  endif
  return  $S$ 
end

```

Algorithm scalar-product(R, E)

begin

 return *< The set obtained by adding R to each element of E >*

end

Algorithm cartesian-product(E)

begin

 return *< The set of all sets that contain one element
 from each of the elements of E >*

end

Appendix D

Domain Theories for the Soybean and DNA Theories

The following two sections present the complete domain theories for the Soybean and DNA domains, respectively.

D.1 Soybean Theory

(diaporthe_stem_canker)	←	(date ?d) (≥ ?d 8) (≤ ?d 9) (high-precip) (stem_cankers above_sec.nde) (fruiting_bodies present) (fruit_pods norm)
(charcoal_rot)	←	(date ?d) (≥ ?d 7) (≤ ?d 8) (low-precip) (high_temp) (plant_growth abnorm) (leaves abnorm) (stem abnorm) (sclerotia present)(roots rotted) (int_discolor black)
(rhizoctonia_root_rot)	←	(date ?d) (≥ ?d 5) (≤ ?d 6) (plant_stand <.normal) (temp <.norm) (precip <.norm) (leaves abnorm) (stem abnorm) (canker_lesion brown) (roots rotted) (hail_canker_relation)
(phytophthora_root_rot)	←	(date ?d) (≥ ?d 4) (≤ ?d 8) (plant_stand <.normal) (date_precip_relation1) (date_temp_relation1) (area_damaged low_areas) (plant_growth abnorm) (leaves abnorm) (stem abnorm) (stem_cankers above_soil) (date_canker_relation) (roots rotted)
(brown_stem_rot)	←	(date ?d) (≥ ?d 7) (≤ ?d 9) (precip >.norm) (low_temp) (leaves abnorm) (stem abnorm) (int_discolor brown) (lodging yes)

(powdery_mildew)	←	(leaves abnorm) (leaf_mild upper_surf)
(downy_mildew)	←	(date ?d) (≥ ?d 6) (≤ ?d 8) (high-precip)
		(area_damaged whole_field) (leaves abnorm)
		(leafspots-halo no_yellow_halos)
		(leaf_mild lower_surf) (date_seed_condition)
		(mold_growth present)
(brown_spot)	←	(leaves abnorm)
		(leafspots-halo no_yellow_halos)
		(leafspots-marg no_w_s_marg)
		(leafspot_size > .1/8)
(bacterial_blight)	←	(date_condition) (date_precip_relation2)
		(date_temp_relation2)
		(leafspots-halo yellow_halos)
		(leafspots-marg w_s_marg)
		(leafspot_size < .1/8) (leaf_shread present)
(bacterial_pustule)	←	(date ?d) (≥ ?d 6) (≤ ?d 8) (high-precip)
		(leaves abnorm)
		(leafspots-halo no_yellow_halos)
		(leafspots-marg no_w_s_marg)
		(leafspot_size < .1/8) (leaf_shread present)
(purple_seed_stain)	←	(date ?d) (≥ ?d 9) (≤ ?d 10) (seed abnorm)
		(seed_discolor present) (seed_size < .norm)
(anthracnose)	←	(date ?d) (≥ ?d 8) (≤ ?d 10) (high-precip)
		(stem abnorm) (canker_lesion brown)
		(fruiting_bodies present)
		(date_seed_condition) (fruit_spot_condition)
(phyllosticta_leaf_spot)	←	(date ?d) (≥ ?d 4) (≤ ?d 7)
		(high-precip) (leaves abnorm)
		(leafspots-halo no_yellow_halos)
		(leafspots-marg no_w_s_marg)
		(leafspot_size > .1/8) (leaf_shread present)
(alternaria_leaf_spot)	←	(date ?d) (≥ ?d 7) (≤ ?d 10)
		(leaves abnorm)
		(leafspots-halo no_yellow_halos)
		(leafspots-marg no_w_s_marg)
		(leafspot_size > .1/8) (leaf_shread absent)
(frog_eye_leaf_spot)	←	(date ?d) (≥ ?d 7) (≤ ?d 9) (high-precip)
		(leaves abnorm)
		(leafspots-halo no_yellow_halos)
		(leafspots-marg no_w_s_marg)
		(leafspot_size > .1/8)

```

(high-precip) ← (precip norm)
(high-precip) ← (precip >_norm)
(low-precip) ← (precip norm)
(low-precip) ← (precip <_norm)
(high-temp) ← (temp norm)
(high-temp) ← (temp >_norm)
(low-temp) ← (temp norm)
(low-temp) ← (temp <_norm)
(hail-canker-relation) ← (hail no) (stem-cankers below_soil)
(hail-canker-relation) ← (hail no) (stem-cankers above_soil)
(hail-canker-relation) ← (hail yes) (stem-cankers above_sec.nde)
(date-precip-relation1) ← (date ?d) (≥ ?d 4) (≤ ?d 6) (precip norm)
(date-precip-relation1) ← (date ?d) (≥ ?d 7) (≤ ?d 8)
                                (precip >_norm)
(date-precip-relation1) ← (date ?d) (> ?d 8)
(date-temp-relation1) ← (date 4) (temp >_norm)
(date-temp-relation1) ← (date ?d) (≥ ?d 5) (≤ ?d 8) (temp norm)
(date-temp-relation1) ← (date ?d) (> ?d 8)
(date-canker-relation) ← (date ?d) (< ?d 5)
(date-canker-relation) ← (date ?d) (> ?d 8)
(date-canker-relation) ← (canker-lesion dk.brown-blk)
(date-seed-condition) ← (date ?d) (< ?d 9)
(date-seed-condition) ← (seed abnorm)
    (date-condition) ← (date ?d) (≥ ?d 4) (≤ ?d 6)
    (date-condition) ← (date ?d) (≥ ?d 8) (≤ ?d 9)
(date-precip-relation2) ← (date ?d) (≥ ?d 4) (≤ ?d 6) (high-precip)
(date-precip-relation2) ← (date ?d) (≥ ?d 8) (≤ ?d 9)
                                (precip >_norm)
(date-precip-relation2) ← (date 7)
(date-precip-relation2) ← (date 10)
(date-temp-relation2) ← (date ?d) (< ?d 8) (temp norm)
(date-temp-relation2) ← (date ?d) (> ?d 8) (temp norm)
(date-temp-relation2) ← (date 8) (temp <_norm) (leaves abnorm)
(fruit-spot-condition) ← (fruit_spots absent)
(fruit-spot-condition) ← (fruit_spots brown.w/blk.specks)
(crop-hist-relation1) ← (crop-hist same_lst.sev.yrs)
(crop-hist-relation1) ← (crop-hist same_lst.two.yrs)
(crop-hist-relation2) ← (crop-hist-relation1)
(crop-hist-relation2) ← (crop-hist same_lst.yr)

```

(damage_date_condition) ← (area_damaged scattered)
 (damage_date_condition) ← (area_damaged low_areas)
 (damage_date_condition) ← (area_damaged upper_areas)
 (damage_date_condition) ← (date 6)
 (damage_date_condition) ← (temp norm)
 (date_temp_relation3) ← (date ?d) (< ?d 6)
 (date_temp_relation3) ← (date ?d) (> ?d 6)
 (date_temp_relation3) ← (temp <_norm)
 (date_pods_condition) ← (date ?d) (< ?d 9)
 (date_pods_condition) ← (fruit_pods diseased)
 (pods_spots_condition) ← (fruit_pods norm)
 (pods_spots_condition) ← (fruit_pods few_present)
 (pods_spots_condition) ← (fruit_pods dna)
 (pods_spots_condition) ← (fruit_spots colored)
 (seed_color_condition) ← (seed norm)
 (seed_color_condition) ← (seed_discolor present)
 (date_spots_condition) ← (date ?d) (< ?d 9)
 (date_spots_condition) ← (date ?d) (> ?d 9)
 (date_spots_condition) ← (fruit_spots colored)

D.2 DNA Theory

(promoter) ← (contact) (conformation)
 (contact) ← (minus_35) (minus_10)
 (minus_35) ← (p-36 t) (p-35 t) (p-34 g) (p-33 a) (p-32 c)
 (minus_35) ← (p-36 t) (p-35 t) (p-34 g) (p-32 c) (p-31 a)
 (minus_10) ← (p-14 t) (p-13 a) (p-12 t) (p-11 a) (p-10 a)
 (p-9 t)
 (minus_10) ← (p-13 t) (p-12 a) (p-10 a) (p-8 t)
 (minus_10) ← (p-12 t) (p-11 a) (p-7 t)
 (conformation) ← (p-47 c) (p-46 a) (p-45 a) (p-43 t) (p-42 t)
 (p-40 a) (p-39 c) (p-22 g) (p-18 t) (p-16 c) (p-8 g)
 (p-7 c) (p-6 g) (p-5 c) (p-4 c) (p-2 c) (p-1 c)
 (conformation) ← (p-45 a) (p-44 a) (p-41 a)
 (conformation) ← (p-49 a) (p-44 t) (p-27 t) (p-22 a) (p-18 t)
 (p-16 t) (p-15 g) (p-1 a)
 (conformation) ← (p-45 a) (p-41 a) (p-28 t) (p-27 t) (p-23 t)
 (p-21 a) (p-20 a) (p-17 t) (p-15 t) (p-4 t)

Appendix E

Detailed Correction Results

This appendix presents detailed examples of EITHER theory modifications for the soybean and DNA theories.

E.1 DNA Theory

The trace in this section was generated by training EITHER using forty training examples.

Summary of EITHER theory changes:

The rule
((minus_35) <- (p-36 t) (p-35 t) (p-34 g) (p-32 c) (p-31 a))
was generalized as follows:
by removing the antecedents,
(p-35 t),
(p-34 g)
to become:
((minus_35) <- (p-36 t) (p-32 c) (p-31 a))

The rule
((minus_10) <- (p-13 t) (p-12 a) (p-10 a) (p-8 t))
was generalized as follows:
by removing the antecedent,
(p-13 t)
to become:
((minus_10) <- (p-12 a) (p-10 a) (p-8 t))

The rule
((minus_10) <- (p-14 t) (p-13 a) (p-12 t) (p-11 a) (p-10 a)
(p-9 t))
was generalized as follows:
by removing the antecedent,
(p-12 t)
to become:
((minus_10) <- (p-14 t) (p-13 a) (p-11 a) (p-10 a) (p-9 t))
by removing the antecedent,
(p-11 a)
to become:
((minus_10) <- (p-14 t) (p-13 a) (p-12 t) (p-10 a) (p-9 t))

The rule
((minus_10) <- (p-12 t) (p-11 a) (p-7 t))
was generalized as follows:
by removing the antecedent,
(p-12 t)
to become:
((minus_10) <- (p-11 a) (p-7 t))
by generalizing the antecedents (p-12 t)(p-7 T)
to become:
((minus_10) <- (p-12 t) (p-11 a) (p-7 a))

```

((minus_10) <- (p-12 a) (p-11 a) (p-7 a))
by generalizing the antecedents (p-12 t)(p-11 A)(p-7 T)
to become:
((minus_10) <- (p-12 t) (p-11 a) (p-7 c))
((minus_10) <- (p-12 t) (p-11 t) (p-7 c))
((minus_10) <- (p-12 a) (p-11 t) (p-7 c))
((minus_10) <- (p-12 c) (p-11 t) (p-7 c))

```

The rule
 ((minus_35) <- (p-36 t) (p-35 t) (p-34 g) (p-33 a) (p-32 c))
 was generalized as follows:

by removing the antecedents,

```
(p-33 a),
```

```
(p-32 c)
```

to become:

```
((minus_35) <- (p-36 t) (p-35 t) (p-34 g))
```

by removing the antecedent,

```
(p-33 a)
```

to become:

```
((minus_35) <- (p-36 t) (p-35 t) (p-34 g) (p-32 c))
```

by removing the antecedent,

```
(p-32 c)
```

to become:

```
((minus_35) <- (p-36 t) (p-35 t) (p-34 g) (p-33 a))
```

by removing the antecedents,

```
(p-36 t),
```

```
(p-33 a),
```

```
(p-32 c)
```

to become:

```
((minus_35) <- (p-35 t) (p-34 g))
```

by removing the antecedent,

```
(p-36 t)
```

to become:

```
((minus_35) <- (p-35 t) (p-34 g) (p-33 a) (p-32 c))
```

by removing the antecedents,

```
(p-35 t),
```

```
(p-32 c)
```

to become:

```
((minus_35) <- (p-36 t) (p-34 g) (p-33 a))
```

by removing the antecedent,

```
(p-35 t)
```

to become:

```
((minus_35) <- (p-36 t) (p-34 g) (p-33 a) (p-32 c))
```

by removing the antecedents,

```
(p-35 t),
```

```
(p-34 g)
```

to become:

```
((minus_35) <- (p-36 t) (p-33 a) (p-32 c))
```

by removing the antecedent,

```
(p-34 g)
```

to become:

```
((minus_35) <- (p-36 t) (p-35 t) (p-33 a) (p-32 c))
```

The rule

```
((promoter) <- (contact) (conformation))
```

had the antecedent

```
(conformation) removed during popup,
```

to become:

```
((promoter) <- (contact))(contact)
```

The generalized rules were reduced using the reduction operators
 to the final set:

```
((promoter) <- (contact))
```

```
((minus_35) <- (intra-0133) (p-33 a) (p-36 t))
```



```

((minus_35) <- (p-36 t) (p-32 c) (p-31 a))
((minus_10) <- (intra-0131) (p-7 c) (p-11 t))
((minus_10) <- (intra-0130) (p-11 a))
((minus_10) <- (p-12 a) (p-10 a) (p-8 t))
((minus_10) <- (inter-0132) (p-12 t) (p-9 t))
((conformation) <- (p-45 a) (p-44 a) (p-41 a))
((inter-0132) <- (p-13 a) (p-14 t) (p-10 a))
((intra-0130) <- (inter-0132) (p-9 t))
((intra-0130) <- (p-12 t) (p-7 a))
((intra-0130) <- (p-12 a) (p-7 a))
((intra-0130) <- (p-12 t) (p-7 c))
((intra-0131) <- (p-12 t))
((intra-0131) <- (p-12 a))
((intra-0131) <- (p-12 c))
((intra-0130) <- (p-7 t))
((intra-0133) <- (p-34 g))
((intra-0133) <- (p-32 c))

```

Final theory:

```

((promoter) <- (contact))
((contact) <- (minus_35) (minus_10))
((minus_35) <- (p-36 t) (p-32 c) (p-31 a))
((minus_35) <- (p-35 t) (p-34 g))
((minus_35) <- (intra-0133) (p-33 a) (p-36 t))
((minus_10) <- (p-12 a) (p-10 a) (p-8 t))
((minus_10) <- (inter-0132) (p-12 t) (p-9 t))
((minus_10) <- (intra-0131) (p-7 c) (p-11 t))
((minus_10) <- (intra-0130) (p-11 a))
((conformation) <- (p-47 c) (p-46 a) (p-45 a) (p-43 t) (p-42 t)
                    (p-40 a) (p-39 c) (p-22 g) (p-18 t) (p-16 c)
                    (p-8 g) (p-7 c) (p-6 g) (p-5 c) (p-4 c) (p-2 c)
                    (p-1 c))
((conformation) <- (p-45 a) (p-44 a) (p-41 a))
((conformation) <- (p-49 a) (p-44 t) (p-27 t) (p-22 a) (p-18 t)
                    (p-16 t) (p-15 g) (p-1 a))
((conformation) <- (p-45 a) (p-41 a) (p-28 t) (p-27 t) (p-23 t)
                    (p-21 a) (p-20 a) (p-17 t) (p-15 t) (p-4 t))
((inter-0132) <- (p-13 a) (p-14 t) (p-10 a))
((intra-0130) <- (inter-0132) (p-9 t))
((intra-0130) <- (p-12 t) (p-7 a))
((intra-0130) <- (p-12 a) (p-7 a))
((intra-0130) <- (p-12 t) (p-7 c))
((intra-0130) <- (p-7 t))
((intra-0131) <- (p-12 t))
((intra-0131) <- (p-12 a))
((intra-0131) <- (p-12 c))
((intra-0133) <- (p-34 g))
((intra-0133) <- (p-32 c))

```

Number of rules: 24

Number of consequents: 9

Number of symbols: 101

Average number of disjuncts: 2.67

Average number of antecedents: 3.21

E.2 Soybean Theory

The theory revision in this section was generated by training EITHER using sixty five training examples.

Summary of EITHER theory changes:

The rule

```
((purple_seed_stain) <- (date ?d) (>= ?d 9) (<= ?d 10)
                                   (seed abnorm) (seed_discolor present)
  (seed_size <_norm))
```

was generalized as follows:

by removing the antecedent,

```
(seed_size <_norm)
```

to become:

```
((purple_seed_stain) <- (date ?d) (>= ?d 9) (<= ?d 10)
                                   (seed abnorm) (seed_discolor present))
```

The rule

```
((hail_canker_relation) <- (hail yes)
                                   (stem_cankers above_sec_nde))
```

was generalized as follows:

by removing the antecedent,

```
(stem_cankers above_sec_nde)
```

to become:

```
((hail_canker_relation) <- (hail yes))
```

The rule

```
((bacterial_pustule) <- (date ?d) (>= ?d 6) (<= ?d 8)
                                   (high_precip) (leaves abnorm)
  (leafspots-halo no_yellow_halos)
  (leafspots-marg no_w-s_marg) (leafspot_size <_1/8)
  (leaf_shread present))
```

was generalized as follows:

by removing the antecedent,

```
(leafspots-halo no_yellow_halos)
```

to become:

```
((bacterial_pustule) <- (date ?d) (>= ?d 6) (<= ?d 8)
                                   (high_precip) (leaves abnorm)
  (leafspots-marg no_w-s_marg)
  (leafspot_size <_1/8)
  (leaf_shread present))
```

The rule

```
((diaporthe_stem_canker) <- (date ?d) (>= ?d 8) (<= ?d 9)
                                   (high_precip)
  (stem_cankers above_sec_nde)
  (fruiting_bodies present)
  (fruit_pods norm))
```

was generalized as follows:

by generalizing the antecedents

```
(date ?d)(>= ?D 8)(<= ?D 9)
```

to become:

```
((diaporthe_stem_canker) <- (date ?d) (>= ?d 7) (<= ?d 10)
                                   (high_precip)
  (stem_cankers above_sec_nde)
  (fruiting_bodies present)
  (fruit_pods norm))
```

The rule

```
((phyllosticta_leaf_spot) <- (date ?d) (>= ?d 4) (<= ?d 7)
                                   (leaves abnorm)
  (leafspots-halo no_yellow_halos)
  (leafspots-marg no_w-s_marg)
  (leafspot_size >_1/8)
  (leaf_shread present))
```

was generalized as follows:

by adding the rule

```
((phyllosticta_leaf_spot) <- (leafspots-halo no_yellow_halos)
    (precip <_norm))
```

The rule

```
((rhizoctonia_root_rot) <- (date ?d) (>= ?d 5) (<= ?d 6)
    (plant_stand <_normal)
    (temp <_norm) (precip <_norm)
    (leaves abnorm)
    (stem abnorm)
    (canker_lesion brown) (roots rotted)
    (hail_canker_relation))
```

was generalized as follows:

by removing the antecedents,

```
(precip <_norm),
(leaves abnorm),
(roots rotted)
```

to become:

```
((rhizoctonia_root_rot) <- (date ?d) (>= ?d 5) (<= ?d 6)
    (plant_stand <_normal)
    (temp <_norm) (stem abnorm)
    (canker_lesion brown)
    (hail_canker_relation))
```

by removing the antecedents,

```
(date ?d),
(>= ?d 5),
(<= ?d 6),
(plant_stand <_normal),
(precip <_norm),
(roots rotted)
```

to become:

```
((rhizoctonia_root_rot) <- (temp <_norm) (leaves abnorm)
    (stem abnorm)
    (canker_lesion brown)
    (hail_canker_relation))
```

The rule

```
((brown_stem_rot) <- (date ?d) (>= ?d 7) (<= ?d 9)
    (precip >_norm) (low_temp)
    (leaves abnorm) (stem abnorm)
    (int_discolor brown) (lodging yes))
```

was generalized as follows:

by removing the antecedent,

```
(precip >_norm)
```

to become:

```
((brown_stem_rot) <- (date ?d) (>= ?d 7) (<= ?d 9) (low_temp)
    (leaves abnorm) (stem abnorm)
    (int_discolor brown) (lodging yes))
```

by removing the antecedents,

```
(precip >_norm),
(leaves abnorm)
```

to become:

```
((brown_stem_rot) <- (date ?d) (>= ?d 7) (<= ?d 9) (low_temp)
    (stem abnorm) (int_discolor brown)
    (lodging yes))
```

The rule

```
((charcoal_rot) <- (date ?d) (>= ?d 7) (<= ?d 8) (low_precip)
    (high_temp) (plant_growth abnorm)
    (leaves abnorm) (stem abnorm)
    (sclerotia present) (roots rotted)
    (int_discolor black))
```

was generalized as follows:

by removing the antecedents,

```
(date ?d),
(>= ?d 7),
(<= ?d 8),
(roots rotted)
```

to become:
 ((charcoal_rot) <- (low_precip) (high_temp)
 (plant_growth abnorm) (leaves abnorm)
 (stem abnorm) (sclerotia present)
 (int_discolor black))

by removing the antecedent,
 (roots rotted)

to become:
 ((charcoal_rot) <- (date ?d) (>= ?d 7) (<= ?d 8)
 (low_precip) (high_temp)
 (plant_growth abnorm) (leaves abnorm)
 (stem abnorm) (sclerotia present)
 (int_discolor black))

The rule

((downy_mildew) <- (date ?d) (>= ?d 6) (<= ?d 8)
 (area_damaged whole_field) (leaves abnorm)
 (leafspots-halo no_yellow_halos)
 (leaf_mild lower_surf)
 (mold_growth present))

was generalized as follows:

by removing the antecedents,

(date ?d),
 (>= ?d 6),
 (<= ?d 8)

to become:

((downy_mildew) <- (area_damaged whole_field)
 (leaves abnorm)
 (leafspots-halo no_yellow_halos)
 (leaf_mild lower_surf)
 (mold_growth present))

by removing the antecedents,

(date ?d),
 (>= ?d 6),
 (<= ?d 8),
 (area_damaged whole_field),
 (leafspots-halo no_yellow_halos)

to become:

((downy_mildew) <- (leaves abnorm) (leaf_mild lower_surf)
 (mold_growth present))

by removing the antecedent,

(area_damaged whole_field)

to become:

((downy_mildew) <- (date ?d) (>= ?d 6) (<= ?d 8)
 (leaves abnorm)
 (leafspots-halo no_yellow_halos)
 (leaf_mild lower_surf)
 (mold_growth present))

The rule

((anthracnose) <- (date ?d) (>= ?d 8) (<= ?d 10)
 (stem abnorm) (canker_lesion brown)
 (fruiting_bodies present)
 (date_seed_condition)
 (fruit_spot_condition))

was generalized as follows:

by removing the antecedent,

(canker_lesion brown)

to become:

((anthracnose) <- (date ?d) (>= ?d 8) (<= ?d 10)
 (stem abnorm) (fruiting_bodies present)
 (date_seed_condition)
 (fruit_spot_condition))

by generalizing the antecedents

(canker_lesion brown)(fruiting_bodies present)

to become:

```
((anthracnose) <- (date ?d) (>= ?d 8) (<= ?d 10)
  (stem abnorm) (canker_lesion brown)
  (date_seed_condition)
  (fruit_spot_condition))
((anthracnose) <- (date ?d) (>= ?d 8) (<= ?d 10)
  (stem abnorm) (canker_lesion dk_brown-blk)
  (date_seed_condition) (fruit_spot_condition))
```

The rule

```
((date_precip_relation1) <- (date ?d) (>= ?d 4) (<= ?d 6)
  (precip norm))
```

was generalized as follows:

by removing the antecedent,

```
(precip norm)
```

to become:

```
((date_precip_relation1) <- (date ?d) (>= ?d 4) (<= ?d 6))
```

The rule

```
((alternaria_leaf_spot) <- (date ?d) (>= ?d 7) (<= ?d 10)
  (leaves abnorm)
  (leafspots-halo no_yellow_halos)
  (leafspots-marg no_w-s_marg)
  (leafspot_size >_1/8)
  (leaf_shread absent))
```

was generalized as follows:

by adding the rules

```
((alternaria_leaf_spot) <- (crop_hist diff_lst_year)
  (fruit_spot_condition)
  (date ?g0191) (>= ?g0191 7.5)
  (leafspot_size >_1/8))
((alternaria_leaf_spot) <- (crop_hist same_lst_yr)
  (fruit_spot_condition)
  (date ?g0191) (>= ?g0191 7.5)
  (leafspot_size >_1/8))
((alternaria_leaf_spot) <- (crop_hist same_lst_two_yrs)
  (fruit_spot_condition)
  (date ?g0191) (>= ?g0191 7.5)
  (leafspot_size >_1/8))
((alternaria_leaf_spot) <- (date ?g0190) (>= ?g0190 9.5)
  (crop_hist same_lst_sev_yrs)
  (fruit_spot_condition)
  (date ?g0191)
  (>= ?g0191 7.5)
  (leafspot_size >_1/8))
```

The rule

```
((phytophthora_root_rot) <- (date ?d) (>= ?d 4) (<= ?d 8)
  (plant_stand <_normal)
  (date_precip_relation1)
  (date_temp_relation1)
  (area_damaged low_areas)
  (plant_growth abnorm)
  (leaves abnorm) (stem abnorm)
  (stem_cankers above_soil)
  (date_canker_relation)
  (roots rotted))
```

was generalized as follows:

by removing the antecedents,

```
(stem_cankers above_soil),
```

```
(roots rotted)
```

to become:

```
((phytophthora_root_rot) <- (date ?d) (>= ?d 4) (<= ?d 8)
  (plant_stand <_normal)
  (date_precip_relation1))
```

```

(date_temp_relation1)
(area_damaged low_areas)
(plant_growth abnorm)
(leaves abnorm) (stem abnorm)
(date_canker_relation))

```

by removing the antecedent,

(roots rotted)

to become:

```

((phytophthora_root_rot) <- (date ?d) (>= ?d 4) (<= ?d 8)
(plant_stand <_normal)
(date_precip_relation1)
(date_temp_relation1)
(area_damaged low_areas)
(plant_growth abnorm)
(leaves abnorm) (stem abnorm)
(stem_cankers above_soil)
(date_canker_relation))

```

The rule

```

((frog_eye_leaf_spot) <- (date ?d) (>= ?d 7) (<= ?d 9)
(high_precip) (leaves abnorm)
(leafspots-halo no_yellow_halos)
(leafspots-marg no_w-s_marg)
(leaf_spot_size >_1/8))

```

was generalized as follows:

by adding the rules

```

((frog_eye_leaf_spot) <- (area_damaged low_areas)
(date_pods_condition) (date ?g0399)
(>= ?g0399 6.5) (germination 80-89%)
(fruit_spots absent))
((frog_eye_leaf_spot) <- (area_damaged upper_areas)
(date_pods_condition) (date ?g0399)
(>= ?g0399 6.5) (germination 80-89%)
(fruit_spots absent))
((frog_eye_leaf_spot) <- (hail_canker_relation)
(fruit_spots colored))

```

The rule

```

((brown_spot) <- (leaves abnorm)
(leafspots-halo no_yellow_halos)
(leafspots-marg no_w-s_marg)
(leafspot_size >_1/8))

```

was generalized as follows:

by adding the rules

```

((brown_spot) <- (high_precip) (date ?g6047) (< ?g6047 6.5)
(leafspot_size >_1/8))
((brown_spot) <- (area_damaged whole_field) (date ?g6046)
(< ?g6046 7.5) (canker_lesion dna)
(date ?g6047) (>= ?g6047 6.5)
(leafspot_size >_1/8))
((brown_spot) <- (canker_lesion brown) (date ?g6047)
(>= ?g6047 6.5) (leafspot_size >_1/8))

```

The rule

```

((bacterial_blight) <- (date_condition)
(date_precip_relation2)
(date_temp_relation2)
(leafspots-halo yellow_halos)
(leafspots-marg w-s_marg)
(leafspot_size <_1/8)
(leaf_shread present))

```

had the antecedent

(date_condition)

removed during popup, to become:

```

((bacterial_blight) <- (date_precip_relation2)
                        (date_temp_relation2)
                        (leafspots-halo yellow_halos)
                        (leafspots-marg w-s_marg)
                        (leafspot_size <_1/8)
                        (leaf_shread present))

The rule
((brown_stem_rot) <- (date ?d) (>= ?d 7) (<= ?d 9) (low_temp)
                    (stem abnorm) (int_discolor brown)
                    (lodging yes))

had the antecedent
(low_temp)
removed during popup, to become:
((brown_stem_rot) <- (date ?d) (>= ?d 7) (<= ?d 9)
                    (stem abnorm) (int_discolor brown)
                    (lodging yes))

The rule
((anthracnose) <- (date ?d) (>= ?d 8) (<= ?d 10)
                  (stem abnorm) (fruiting_bodies present)
                  (date_seed_condition)
                  (fruit_spot_condition))

had the antecedent
(date_seed_condition)
removed during popup, to become:
((anthracnose) <- (date ?d) (>= ?d 8) (<= ?d 10)
                  (stem abnorm) (fruiting_bodies present)
                  (fruit_spot_condition))

The rule
((anthracnose) <- (date ?d) (>= ?d 8) (<= ?d 10)
                  (stem abnorm) (canker_lesion brown)
                  (date_seed_condition)
                  (fruit_spot_condition))

had the antecedent
(date_seed_condition)
removed during popup, to become:
((anthracnose) <- (date ?d) (>= ?d 8) (<= ?d 10)
                  (stem abnorm) (canker_lesion brown)
                  (fruit_spot_condition))

The rule
((anthracnose) <- (date ?d) (>= ?d 8) (<= ?d 10)
                  (stem abnorm) (canker_lesion dk_brown-blk)
                  (date_seed_condition)
                  (fruit_spot_condition))

had the antecedent
(date_seed_condition)
removed during popup, to become:
((anthracnose) <- (date ?d) (>= ?d 8) (<= ?d 10)
                  (stem abnorm) (canker_lesion dk_brown-blk)
                  (fruit_spot_condition))

The rule
((downy_mildew) <- (date ?d) (>= ?d 6) (<= ?d 8)
                  (area_damaged whole_field) (leaves abnorm)
                  (leafspots-halo no_yellow_halos)
                  (leaf_mild lower_surf)
                  (date_seed_condition)
                  (mold_growth present))

had the antecedent
(date_seed_condition)
removed during popup, to become:
((downy_mildew) <- (date ?d) (>= ?d 6) (<= ?d 8)
                  (area_damaged whole_field) (leaves abnorm)
                  (leafspots-halo no_yellow_halos)
                  (leaf_mild lower_surf)
                  (mold_growth present))

```

The rule

```
((phyllosticta_leaf_spot) <- (date ?d) (>= ?d 4) (<= ?d 7)
    (high_precip) (leaves abnorm)
    (leafspots-halo no_yellow_halos)
    (leafspots-marg no_w-s_marg)
    (leafspot_size >_1/8)
    (leaf_shread present))
```

had the antecedent

(high_precip)

removed during popup, to become:

```
((phyllosticta_leaf_spot) <- (date ?d) (>= ?d 4) (<= ?d 7)
    (leaves abnorm)
    (leafspots-halo no_yellow_halos)
    (leafspots-marg no_w-s_marg)
    (leafspot_size >_1/8)
    (leaf_shread present))
```

The rule

```
((downy_mildew) <- (date ?d) (>= ?d 6) (<= ?d 8)
    (high_precip) (area_damaged whole_field)
    (leaves abnorm)
    (leafspots-halo no_yellow_halos)
    (leaf_mild lower_surf)
    (date_seed_condition)
    (mold_growth present))
```

had the antecedent

(high_precip)

removed during popup, to become:

```
((downy_mildew) <- (date ?d) (>= ?d 6) (<= ?d 8)
    (area_damaged whole_field) (leaves abnorm)
    (leafspots-halo no_yellow_halos)
    (leaf_mild lower_surf)
    (date_seed_condition)
    (mold_growth present))
```

The rule

```
((anthracnose) <- (date ?d) (>= ?d 8) (<= ?d 10)
    (high_precip) (stem abnorm)
    (canker_lesion brown)
    (fruiting_bodies present)
    (date_seed_condition)
    (fruit_spot_condition))
```

had the antecedent

(high_precip)

removed during popup, to become:

```
((anthracnose) <- (date ?d) (>= ?d 8) (<= ?d 10)
    (stem abnorm) (canker_lesion brown)
    (fruiting_bodies present)
    (date_seed_condition)
    (fruit_spot_condition))
```

The rule

```
((phytophthora_root_rot) <- (date ?d) (>= ?d 4) (<= ?d 8)
    (plant_stand <_normal)
    (date_precip_relation1)
    (date_temp_relation1)
    (area_damaged low_areas)
    (plant_growth abnorm)
    (leaves abnorm) (stem abnorm)
    (date_canker_relation))
```

had the antecedent

(date_temp_relation1)

removed during popup, to become:

```
((phytophthora_root_rot) <- (date ?d) (>= ?d 4) (<= ?d 8)
    (plant_stand <_normal)
```



```

(date_precip_relation1)
(area_damaged low_areas)
(plant_growth abnorm)
(leaves abnorm) (stem abnorm)
(date_canker_relation))

```

The generalized rules were reduced using the reduction operators to the final set:

```

((diaporthe_stem_canker) <- (date ?d) (>= ?d 7) (<= ?d 10)
  (high_precip)
  (stem_cankers above_sec_nde)
  (fruiting_bodies present)
  (fruit_pods norm))
((charcoal_rot) <- (low_precip) (high_temp)
  (plant_growth abnorm) (leaves abnorm)
  (stem abnorm) (sclerotia present)
  (int_discolor black))
((rhizoctonia_root_rot) <- (date ?d) (>= ?d 5) (<= ?d 6)
  (plant_stand <_normal) (temp <_norm)
  (stem abnorm) (canker_lesion brown)
  (hail_canker_relation))
((rhizoctonia_root_rot) <- (temp <_norm) (leaves abnorm)
  (stem abnorm) (canker_lesion brown)
  (hail_canker_relation))
((phytophthora_root_rot) <- (date ?d) (>= ?d 4) (<= ?d 8)
  (plant_stand <_normal)
  (date_precip_relation1)
  (area_damaged low_areas)
  (plant_growth abnorm)
  (leaves abnorm) (stem abnorm)
  (date_canker_relation))
((brown_stem_rot) <- (date ?d) (>= ?d 7) (<= ?d 9)
  (stem abnorm) (int_discolor brown)
  (lodging yes))
((downy_mildew) <- (leaves abnorm) (leaf_mild lower_surf)
  (mold_growth present))
((brown_spot) <- (inter-2643) (leafspot_size >_1/8))
((brown_spot) <- (high_precip) (date ?g6047) (< ?g6047 6.5)
  (leafspot_size >_1/8))
((brown_spot) <- (intra-2645) (date ?g6047)
  (>= ?g6047 6.5))
((bacterial_blight) <- (date_precip_relation2)
  (date_temp_relation2))
(leafspots-halo yellow_halos)
  (leafspots-marg w-s_marg)
  (leafspot_size <_1/8)
(leaf_shread present))
((bacterial_pustule) <- (date ?d) (>= ?d 6) (<= ?d 8)
  (high_precip) (leaves abnorm)
  (leafspots-marg no_w-s_marg)
  (leafspot_size <_1/8)
  (leaf_shread present))
((purple_seed_stain) <- (date ?d) (>= ?d 9) (<= ?d 10)
  (seed abnorm) (seed_discolor present))
((anthracnose) <- (intra-2642) (fruit_spot_condition)
  (stem abnorm) (date ?d) (>= ?d 8) (<= ?d 10))
((phyllosticta_leaf_spot) <- (inter-2643) (date ?d) (>= ?d 4)
  (<= ?d 7) (leafspot_size >_1/8)
  (leaf_shread present))
((phyllosticta_leaf_spot) <- (leafspots-halo no_yellow_halos)

```

```

      (precip <_norm))
((alternaria_leaf_spot) <- (intra-2641) (leafspot_size >_1/8)
      (fruit_spot_condition) (date ?g0191)
      (>= ?g0191 7.5))
((alternaria_leaf_spot) <- (inter-2643) (date ?d) (>= ?d 7)
      (<= ?d 10) (leafspot_size >_1/8)
      (leaf_shread absent))
((frog_eye_leaf_spot) <- (intra-2644) (fruit_spots absent)
      (germination 80-89%)
      (date_pods_condition) (date ?g0399)
      (>= ?g0399 6.5))
((frog_eye_leaf_spot) <- (inter-2643) (date ?d) (>= ?d 7)
      (<= ?d 9) (high_precip)
      (leafspot_size >_1/8))
((frog_eye_leaf_spot) <- (hail_canker_relation)
      (fruit_spots colored))
((high_precip) <- (precip norm))
((low_temp) <- (temp norm))
((hail_canker_relation) <- (hail yes))
((date_precip_relation1) <- (date ?d) (>= ?d 4) (<= ?d 6))
((date_temp_relation1) <- (date ?d) (> ?d 8))
((date_seed_condition) <- (date ?d) (< ?d 9))
((date_condition) <- (date ?d) (>= ?d 8) (<= ?d 9))
((inter-2643) <- (leafspots-halo no_yellow_halos)
      (leafspots-marg no_w-s_marg) (leaves abnorm))
((intra-2642) <- (fruiting_bodies present))
((intra-2642) <- (canker_lesion brown))
((intra-2642) <- (canker_lesion dk_brown-blk))
((intra-2641) <- (crop_hist diff_lst_year))
((intra-2641) <- (crop_hist same_lst_yr))
((intra-2641) <- (crop_hist same_lst_two_yrs))
((intra-2641) <- (date ?g0190) (>= ?g0190 9.5)
      (crop_hist same_lst_sev_yrs))
((intra-2644) <- (area_damaged low_areas))
((intra-2644) <- (area_damaged upper_areas))
((intra-2645) <- (area_damaged whole_field) (date ?g6046)
      (< ?g6046 7.5) (canker_lesion dna)
      (leafspot_size >_1/8))
((intra-2645) <- (canker_lesion brown) (leafspot_size >_1/8))

```

The rule

```

((diaporthe_stem_canker) <- (date ?d) (>= ?d 7) (<= ?d 10)
      (high_precip)
      (stem_cankers above_sec_ndc)
      (fruiting_bodies present)
      (fruit_pods norm))

```

was specialized by replacing it with the following inductively learned rules:

```

((diaporthe_stem_canker) <- (date ?d) (>= ?d 7) (<= ?d 10)
      (high_precip)
      (stem_cankers above_sec_ndc)
      (fruiting_bodies present)
      (fruit_pods norm)
      (date_temp_relation2))

```

The rule

```

((fruit_spot_condition) <- (fruit_spots absent))

```

was specialized by replacing it with the following inductively learned rules:

```

((fruit_spot_condition) <- (fruit_spots absent) (intra-2641))

```

Final theory:

```

((diaporthe_stem_canker) <- (date ?d) (>= ?d 7) (<= ?d 10)

```

```

        (high_precip)
        (stem_cankers above_sec_nde)
        (fruiting_bodies present)
        (fruit_pods norm)
        (date_temp_relation2))
((charcoal_rot) <- (low_precip) (high_temp)
  (plant_growth abnorm) (leaves abnorm)
  (stem abnorm) (sclerotia present)
  (int_discolor black))
((rhizoctonia_root_rot) <- (date ?d) (>= ?d 5) (<= ?d 6)
  (plant_stand <_normal) (temp <_norm)
  (stem abnorm) (canker_lesion brown)
  (hail_canker_relation))
((rhizoctonia_root_rot) <- (temp <_norm) (leaves abnorm)
  (stem abnorm) (canker_lesion brown)
  (hail_canker_relation))
((phytophthora_root_rot) <- (date ?d) (>= ?d 4) (<= ?d 8)
  (plant_stand <_normal)
  (date_precip_relation1)
  (area_damaged low_areas)
  (plant_growth abnorm)
  (leaves abnorm) (stem abnorm)
  (date_canker_relation))
((brown_stem_rot) <- (date ?d) (>= ?d 7) (<= ?d 9)
  (stem abnorm) (int_discolor brown)
  (lodging yes))
((powdery_mildew) <- (leaves abnorm) (leaf_mild upper_surf))
((downy_mildew) <- (leaves abnorm) (leaf_mild lower_surf)
  (mold_growth present))
((brown_spot) <- (high_precip) (date ?g6047) (< ?g6047 6.5)
  (leafspot_size >_1/8))
((brown_spot) <- (inter-2643) (leafspot_size >_1/8))
((brown_spot) <- (intra-2645) (date ?g6047) (>= ?g6047 6.5))
((bacterial_blight) <- (date_precip_relation2)
  (date_temp_relation2)
  (leafspots-halo yellow_halos)
  (leafspots-marg w-s_marg))
(leafspot_size <_1/8) (leaf_shread present))
((bacterial_pustule) <- (date ?d) (>= ?d 6) (<= ?d 8)
  (high_precip) (leaves abnorm)
  (leafspots-marg no_w-s_marg)
  (leafspot_size <_1/8)
  (leaf_shread present))
((purple_seed_stain) <- (date ?d) (>= ?d 9) (<= ?d 10)
  (seed abnorm) (seed_discolor present))
((anthracnose) <- (intra-2642) (fruit_spot_condition)
  (stem abnorm) (date ?d) (>= ?d 8) (<= ?d 10))
((phyllosticta_leaf_spot) <- (inter-2643) (date ?d) (>= ?d 4)
  (<= ?d 7) (leafspot_size >_1/8)
  (leaf_shread present))
((phyllosticta_leaf_spot) <- (leafspots-halo no_yellow_halos)
  (precip <_norm))
((alternaria_leaf_spot) <- (inter-2643) (date ?d) (>= ?d 7)
  (<= ?d 10) (leafspot_size >_1/8)
  (leaf_shread absent))
((alternaria_leaf_spot) <- (intra-2641) (leafspot_size >_1/8)
  (fruit_spot_condition) (date ?g0191)
  (>= ?g0191 7.5))
((frog_eye_leaf_spot) <- (inter-2643) (date ?d) (>= ?d 7)
  (<= ?d 9) (high_precip)

```

VITA

Dirk Ourston was born in Los Angeles, California, on November 21, 1942, the son of Karl Ourston and Beulah Marjorie Ourston. He attended El Monte Union High School in El Monte, California, from September 1956 until June 1960. He attended California State University at Los Angeles from September 1960 until June 1964, receiving the degree of Bachelor of Science in Physics. He attended the University of California at Los Angeles from September 1966 until December 1968, and received the degree of Master of Science in Engineering. From 1964 until 1989 he was employed in various technical and management positions, primarily within the aerospace industry. In 1989 he began work as a research assistant at the University of Texas. He was admitted to the Graduate School of the University of Texas in the fall of 1986.

Permanent address: 2713 Charlesworth Dr.
Austin, Texas 78745

This dissertation was typeset¹ with \LaTeX by the author.

¹ \LaTeX document preparation system was developed by Leslie Lamport as a special version of Donald Knuth's \TeX program for computer typesetting. \TeX is a trademark of the American Mathematical Society. The \LaTeX macro package for The University of Texas at Austin dissertation format was written by Khe-Sing The.