Extraction and Use of Contextual Attributes for Theory Completion: An Integration of Explanation-Based and Similarity-Based Learning

Andrea Pohoreckyj Danyluk

CUCS-005-92

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences.

Columbia University
1992

ABSTRACT

Extraction and Use of Contextual Attributes for Theory Completion: An Integration of Explanation-Based and Similarity-Based Learning

Andrea Pohoreckyj Danyluk

This research investigates the use of contextual cues to address problems in machine learning that arise from assumptions about the initial knowledge that is necessary for the acquisition of new information. Machine learning approaches may be placed along a spectrum describing purely inductive to purely deductive techniques. Inductive systems possess essentially no explicit knowledge that can be used in acquiring new facts, while deductive systems are assumed to contain a complete theory of the domain. Most work in machine learning has concentrated on approaches at the two ends of the spectrum. This dissertation describes an approach that integrates inductive and deductive methods. It provides a mechanism by which induction can be used in order to detect and acquire knowledge missing from the domain theory of a deductive system, while extracting information to guide the inductive search for missing knowledge. This information may be termed *contextual*.

Much knowledge can be brought to bear in filling gaps in a deductive system's domain theory. This might include information about the domain theory itself or perhaps the history of the learning system. I show that a domain-independent set of attributes can be used to describe contextual information available in the deductions made by a system. Furthermore, I present an algorithm for using that information to generate domain-independent heuristics that guide induction. The heuristics use context information to select examples for induction of missing information and guide the way the examples are interpreted. My empirical investigations have yielded a characterization of the efficacy of various subsets of attributes.

The algorithm for extracting and applying contextual knowledge is implemented in the Gemini system. Gemini combines similarity-based learning, an inductive technique, and explanation-based learning, a deductive method, in an architecture of mutual dependence. Similarity-based learning is used to fill gaps detected in the domain theory used by explanation-based learning. I have developed Gemini as a general learning system and have tested it in four domains. It has been the testbed for my investigation of contextual attributes and confirms that there exists a set of domain-independent attributes that can effectively guide an inductive search for concept definitions.

Acknowledgements

This dissertation would never have been completed without the help and encouragement of many people.

First and foremost I owe many thanks to my two advisors, Kathy McKeown and Mike Lebowitz. Mike always understood what I was doing before I understood it myself, while Kathy forced me to work out the details. They both pushed me when I needed a push, and had the patience to leave me alone when I needed time to work on something. Most importantly, though, they were both incredibly encouraging.

I was fortunate to have wonderful committee members. Terry Boult met with me on a number of occasions to help me work out my research methodology. He, as well as Steve Feiner, Jim Corter, and Paul Benjamin, gave me great feedback on the dissertation itself. They did everything, from raising hard questions to suggesting changes to the writing.

My officemates throughout the years — Nasser Barghouti, Tom Ellman, Nihal Nounou, Nat Polish, and Tony Weida at Columbia and John Martin and Gary Sevitsky at NYNEX — have had to put up with my endless frustrations and complaints. They've all been wonderfully patient and supportive. Tony, Nat, John, and Gary were especially kind in dealing with my whining during the last few months before my defense.

Germaine L'Eveque helped to make navigation through the Columbia bureaucracy as uncomplicated as possible. Without her I would still be trying to figure out what forms to file.

And last, but not least, my friends both in the department and outside gave me the confidence to keep going. A couple were especially wonderful at times that were most difficult for me. Dannie Durand was a great friend when I was going through one of my greatest crisis periods. Ursula Wolz was my ‘‘companion in misery’’, jumping the hurdles with me and reminding me that I could do it. My friends and family made the last seven years not only bearable, but fun.

This research was supported in part by the Defense Advanced Research Projects Agency under contract N00039-84-C-0165.

To Andrew

Chapter 1

Introduction

1.1. An Overview of the Dissertation

Artificial Intelligence (AI) problem solvers often make use of explicitly encoded knowledge about their domain of application. In general the knowledge is assumed to be complete and correct. This is clearly unrealistic in all but the simplest domains. In order for AI systems to be truly autonomous, they must be able to adapt in situations where their knowledge is less than adequate. The field of machine learning has largely grown out of the desire to solve this problem. This dissertation addresses the use of machine learning techniques to solve the problem of detecting and correcting incompleteness in a knowledge base.

Two disparate machine learning techniques bear on the task of completing partial knowledge bases. These are explanation-based and similarity-based learning. Explanation-based learning (EBL) is a technique where the system's goal is to reformulate an existing knowledge base to make it more efficient. More specifically, the basic goal may be to more efficiently recognize concepts that the system is already capable of recognizing. The EBL system begins with a complete and consistent knowledge base for a particular domain (called the *domain theory*). Given an environment-provided example of a specific domain concept, the system uses the domain theory to extract the concept's characteristic features. The system's analysis proceeds by using the domain theory to generate an explanation describing why the instance is a member of the concept to be learned. While EBL has been applied with success to a variety of problems, it suffers from the same assumptions made about AI knowledge bases in general. The method assumes that the domain theory is complete, correct, and tractable to use.

Similarity-based learning (SBL), on the other hand, requires essentially no initial knowledge. Its goal is to acquire descriptions that will allow the system to recognize concepts it does not yet know. Many examples of the concept are given to the system. These are compared against each other in various ways to find a definition of the concept. SBL suffers because of its lack of an explicit domain theory, however. The method requires that human intervention play a large role in tailoring input examples to allow the system to converge on a good concept definition.

This dissertation addresses the problem of incompleteness in a domain theory. Rather than considering all AI problem solving systems, we focus on incompleteness in explanation-based learning. Our solution is based on using SBL to learn concept definitions missing from an EBL domain theory. However, we also introduce heuristics that exploit contextual information from previously derived EBL explanations in order to learn the missing knowledge. Thus a second problem addressed is the use of existing domain knowledge to guide learning performed by SBL.

Our method has largely been motivated by the desire to eliminate restrictive requirements of earlier approaches. Specifically, we have concentrated on devising an approach that maximizes the autonomy of the learning system. We show that our contextual heuristics that guide learning make use of *generally available* explanatory knowledge, and that they can therefore be machine generated rather than hand-crafted for each domain. We also introduce a mechanism by which information may be efficiently stored by the learning system for later use. We give an algorithm for organizing input examples into a taxonomy that describes the examples' structural and explanatory similarities. When examples are needed for the creation of missing knowledge, they may be automatically extracted from this taxonomy.

In order to demonstrate the efficacy and generality of our approach, we have implemented Gemini, a learning system that combines EBL and SBL in a feedback loop. When a gap is detected in the EBL domain theory, Gemini's SBL component induces the missing information. It makes use of previously seen examples, which it has stored into the taxonomy described. It also makes use of domain-independent contextual heuristics that exploit any existing explanatory knowledge. Gemini has been applied to four domains: network fault diagnosis, two domains describing military radio fault diagnosis, and terrorist event news stories. These domains span a wide range of classification tasks.

In the next section we give additional background information describing the two machine learning approaches that are the focus of our work: explanation-based and similarity-based learning. After that we give an example to highlight the major problems we have addressed. We then discuss our solutions to the problems. We highlight the contributions of the research, contrasting our work with earlier approaches. We conclude this chapter with a reader's guide to the dissertation.

1.2. Background

1.2.1. Machine Learning as a Knowledge Engineering Tool: Historical Perspective

Much recent work in machine learning has grown out of expert systems research. During the development of early expert systems, it was recognized that the task of knowledge acquisition is a formidable one. In order to facilitate this process a number of tools were developed: (1) knowledge base editors and interfaces, (2) explanation facilities, and (3) knowledge base revision facilities.[1]

Knowledge acquisition tools provided sophisticated mechanisms for the building and revision of expert system knowledge bases. TEIRESIAS [Davis 76], for example, allowed the expert to specify a new rule, or rule revision, in natural language format. It interacted with the user to check for consistency between the concepts expressed in the rule and those already represented in the knowledge base. However, the dependence upon interaction with the expert remained a bottleneck in the knowledge acquisition process. Much work in machine learning grew out of the desire to further automate the building of knowledge bases.

[1]From [Building Expert Systems 83].

1.2.2. Similarity-Based and Explanation-Based Learning

There are two disparate tasks that bear on knowledge acquisition. The first is the task of automating the acquisition of new knowledge. The second involves improving performance through the reformulation or reorganization of currently held knowledge. Two classes of machine learning mechanisms have grown out of the desire to address these tasks. Similarity-based learning (SBL) is an inductive learning approach that generally involves the analysis of large quantities of data to formulate general rules. Explanation-based learning (EBL) involves generating explanations (or deductions) that are then the basis for condensing and reformulating the knowledge expressed within them.

Similarity-based learning takes as its input large quantities of data from which a generalization is to be learned. The SBL system considers the data in order to extract characteristic features that define a class of objects. The system may be asked to separate its inputs into subsets based upon their similarities or differences. Alternately, it may be given the name of a specific concept to be learned. If a specific single concept is to be learned, the input data may be labeled as positive examples of the concept or as negative examples.

Explanation-based learning begins with a domain theory, often expressed as a rule base, that it must reformulate to make it more useful or efficient. Input to EBL is the name of a concept (termed the *goal concept*) and an instance of that concept (the *input example*). Using the domain theory, EBL generates an explanation showing that the instance is an example of the concept to be learned. Often, the explanation can be represented as a *tree*, with the goal concept at the root, and actual input example features at the leaves. Internal nodes (or *subgoals*) are intermediate conclusions drawn from the input using the domain theory.

After EBL generates an explanation, it generalizes the instance such that an analogous generalization of the explanation would still hold. The generalized instance can be used directly by the system to infer concept membership, rather than requiring that the system re-derive the explanation using the domain theory, hopefully improving efficiency. The derived explanation can also be stored for later use.

Both EBL and SBL have been applied with success to a number of problems. The two approaches are not without shortcomings, however. EBL's shortcomings stem from its knowledge intensive nature. Like the expert systems for which machine learning is needed, EBL assumes that its domain theory is complete, correct, and that the derivation of explanations using the theory is tractable. The problems of SBL stem largely from its ''knowledge poor'' nature. Because SBL has no domain theory for guidance, it must rely on large amounts of data for accuracy. This is problematic if inputs are not readily available. Some SBL techniques can make use of less data if positive and negative input examples are carefully labeled by an outside source. This is problematic if the task of learning is to minimize the need for a human expert.

1.3. An Example of the Problems Addressed

1.3.1. Network Fault Diagnosis

One domain that we have used to test the ideas presented in the dissertation is network fault diagnosis. The network in this domain is an ethernet/token ring network. A fault in the network is signalled by the inability of one user to reach another user connected to the same network. The knowledge used by the performance system is encoded on a level that enables isolation of the fault to a particular segment of the network, but does not allow deeper analysis of the problem. A sample rule for this domain is:

```
IF
    HAS (?X, COMPLAINT-USER-CANT-CONNECT?, T) ∧
    HAS (?X, CAN-USER-REACH-ANOTHER-NET?, T)

THEN
    ASSERT (LOOKS-LIKE-DEST-PROB)
```

which states that ''If a user is complaining that a connection cannot be established with a particular destination, but the same user is able to reach other nodes on the network, then the problem lies with the destination of the unestablished connection.'' The rule base has been encoded from a prototype knowledge base that was extracted from experts maintaining the CMU campus computer network [Eshelman 88].

Examples will be taken from the network fault diagnosis domain to illustrate points in the following section. Examples from our other three domains will be used in later chapters, where the domains will be described in greater detail.

1.3.2. Incompleteness in the Network Fault Domain Theory

Input to EBL, as described earlier, is the name of a concept to be learned and an instance of that concept. In the network fault diagnosis domain, suppose the goal concept is the diagnosis *poss-user-wrkstn->net-conn-prob*, a possible problem in the connection of the user workstation to the network. Next, suppose that the example given to the system as an instance of the problem is represented as the frame shown in Figure 1. Call this input fault81.

Given this input, the EBL system uses its domain theory to generate an explanation showing that the example is an instance of the given concept. An explanation for this input is shown in Figure 2.

If the domain theory is incomplete, however, the EBL system may only be able to generate the partial explanation shown in Figure 3. This might occur for one of two reasons:

1. The domain theory might contain *no* rule in which the concept *looks-like-source-prob* is concluded.
2. The domain theory might contain rules in which *looks-like-source-prob* is concluded, but those rules may be too specific to apply to the input example.

In either case, the knowledge base must be augmented.

We can see from the explanation in Figure 2 that the rule needed to complete the explanation of Figure 3 should link all or some of the features of the input directly to the missing concept. That is, the missing concept can be defined by a conjunction of input features, as shown in Figure 4. Rather than simply learning a rule that will adequately cover the current situation, our task is to learn rules that can be applied reliably to future

Figure 1: Hypothetical Input to EBL
for Network Fault Diagnosis

examples. Thus we define the incomplete theory problem to be the problem of inducing missing concept definitions.

In general, one cannot assume that missing rules will conveniently fall at the end of an explanation, where input example features can be directly linked to the unproven concept. We extend the problem of learning new rules to include:

1. Learning rules missing ‘‘in the middle’’ of an explanation. The problem is still defined as inductive concept learning, but requires additional work to replace input features by inferences that can be derived from them.
2. Learning multiple missing rules from a single explanation.

Figure 2: Explanation Generated by EBL
for the Network Fault Diagnosis Example

Our solution to the problem of incompleteness is based on similarity-based induction of missing concepts. The rule, or concept definition, learned is not learned in isolation, but rather is learned in the *context* of a partial explanation. We will show that useful information can be extracted from partial explanations (as well as earlier complete ones), in order to *bias* the manner in which induction proceeds. Thus a second major problem addressed in this dissertation is that of finding a mechanism to explicitly bias (or guide) similarity-based learning.

Figure 3: Explanation Generated by EBL
When the Theory is Incomplete

Figure 4: Linking Input Features to
an Unproven Concept

1.4. An Introduction to the Solutions

This section outlines our solutions to the two problems addressed:

1. The augmentation of incomplete domain theories;
2. The identification of knowledge that can explicitly be exploited to guide induction of knowledge to augment a domain theory.

1.4.1. Problem I: Completing an Overly Specific Domain Theory

As described earlier, the incomplete domain theory problem can be viewed as induction of missing concept definitions. The basis for our solution to this problem is similarity-based learning. This choice stems from our objective of creating a solution that will allow the system to be as autonomous as possible. Though examples are required for induction by SBL, the method itself does not require substantial auxiliary domain knowledge. Our assumption is that, ultimately, inductive/empirical methods must play a major role in learning knowledge to complete a domain theory.

Our problem differs, however, from the standard problem of inductive learning. Our concept definitions are not learned in isolation, but rather must be integrated into an existing theory. Because a theory (albeit partial) exists, however, the learning system has access to additional information that may be used in creating new rules. The partial explanation sets the *context* in which the new rule is to be learned. Contextual information may be used to influence the way induction proceeds. For example, it might be reasonable to assume that features are not generally referenced multiple times within a single explanation. Thus those already appearing in a partial explanation might be eliminated from consideration for the definition of the missing subgoal.

More generally, useful information may be extracted from both current and past explanations (partial or complete) to guide the induction. We will describe information that is generally available in EBL explanations and will show how it forms a basis for generating contextual heuristics. We give an algorithm for performing the generation, and we evaluate a subset of the heuristics thus created.

1.4.2. Problem II: Using Knowledge to Bias Inductive Learning

Similarity-based learning plays an important role in our solution to domain theory incompleteness. As already introduced, however, SBL suffers from problems related to the fact that it makes use of little explicit knowledge. To compensate for its lack of domain knowledge, SBL requires large numbers of input examples. And while it is praised for requiring no explicit knowledge, it does make use of implicit knowledge, such as the ordering of input examples and their representation. Though there is much implicit knowledge that biases the induction of new knowledge, it is rarely evaluated because it is difficult to isolate explicitly. We show that one class of knowledge — contextual information from EBL's explanations — can be explicitly encoded. We consider a subset of that information and evaluate its effect on learning.

More specifically, we show that, when contextual knowledge exists, it can be used as an important aid in labeling instances as positive or negative for concept learning, so that an outside source need not do so. We also show that the domain knowledge can be used to generalize learned rules when large numbers of examples are not available.

1.4.3. A Preview of Contextual Heuristics

Contextual heuristics that exploit existing domain knowledge are a central contribution of our research. They can be used in two ways:

1. To label examples as positive or negative for induction.
2. To generalize new rules and thus extend their applicability.

In performing induction of a missing concept, a system must have access to positive and/or negative examples of that concept. Given our motivation to make the learning system autonomous, we cannot assume that an external source will be available to provide the learning system with additional appropriate examples. As will be shown in Chapter 3, however, examples seen by a system can be efficiently stored to be retrieved later when a new rule is required. The examples will not necessarily have been labeled as positive or negative instances of the missing concept though. We claim that their concept membership can be inferred by considering their relationship to a known positive example (the instance for which only a partial explanation could be generated). Consider the following heuristic:

> *If a previously seen example had an explanation significantly different from the current partial explanation, treat it as a negative example of the concept to be learned.*

This heuristic assumes that because the analysis (explanation) of the earlier example was so different from the current one, the earlier example must *not* be a member of the concept to be learned for the current one. The examples simply differ too greatly to overlap in any concept membership. Negative examples are useful for induction as they allow a system to focus on features that help to distinguish one concept definition from another. Say a concept definition is to be learned. We want that concept definition to be sufficiently general to encompass all instances of that concept. Yet it must not be so general as to classify examples of other concepts. Negative examples allow the inductive learning system to tailor learned definitions to ensure that they will not be too broad.

In the case of a partial domain theory, we want to ensure this behavior as well. When a new rule — the definition of a missing subgoal — is required, it should not be made so broad as to apply in situations where it should not. If there is no outside source to present negative examples for learning, the above heuristic would be useful. It says that a good rule of thumb to follow is to be sure that the new concept definition does not cover examples whose explanations differ significantly from the one to be completed. Presumably the explanations provide a reasonably reliable analysis of the examples.

In addition to using *labeled* examples, SBL systems may require a *large* set of examples from which the new concept is induced. It is not often sufficient to receive a small set of of appropriately labeled positive and negative examples. A large example base might not be readily available, however. When large numbers of examples cannot be found, contextual information may sometimes be used to modify a concept definition. For example, consider the following heuristic, mentioned earlier:

> *If a feature already appears in the partial explanation, assume that it will not appear multiple times, and remove it from the definition of the missing concept.*

This heuristic generalizes the new definition by eliminating features (or conditions) for applicability. It does so without the aid of additional examples.

1.5. Implementation and Evaluation of Solutions: Gemini

Our solution to the incomplete theory problem applies heuristics that make use of contextual information to label previously seen examples as positive or negative for induction and to generalize the rules learned from those examples beyond the generalization indicated by the examples themselves. In order to demonstrate the applicability and generality of our approach, we have implemented a system called Gemini, which we have tested in four domains.

Gemini integrates EBL and SBL in a feedback loop, allowing intercommunication for the mutual benefit of the two methods. Input first enters the EBL component of the system, which uses a built-in domain theory to generate explanations. If a complete explanation can be generated, the input, along with its explanation, is stored by the SBL component for later use. It might be of value in the future, when a missing concept definition is required. The example is organized into a taxonomy that describes similarities and differences among examples.

If only a partial explanation can be derived by EBL, the SBL component is called upon to provide the rule(s) that are missing. In order to do so, it retrieves relevant examples from the taxonomy of inputs it has previously created. It does so by invoking a number of contextual heuristics that indicate what types of examples might best be used as positive and negative instances for induction. The learned rule is later placed in the EBL domain theory. The learned rule is treated as any previously defined rule. It is assumed to be correct, and thus it cannot be retracted. It can, however, be made superfluous by the later addition of a more general (i.e., more complete) rule.

Since our contextual heuristics are rules of thumb, they will not all be equally useful. Nor will they be equally useful in all domains. We verify the efficacy of contextual heuristics empirically using Gemini. More generally, we have conducted tests to characterize the applicability of a subset of heuristics in four different domains.

1.6. Contributions to Machine Learning

The following is a summary and discussion of our contributions to machine learning. We divide the contributions into two major groups: those advancing EBL and those for SBL. For each we discuss the improvements we have made over earlier solutions to the problems addressed.

1.6.1. Contributions to EBL

The principal problem we have addressed in the context of explanation-based learning is the problem of domain theory incompleteness. That is, we have addressed the problem of augmenting an overly specific domain theory. Our solution is based on SBL, but uses contextual information to guide the induction.

Our work is not the first to address this problem. Previous approaches, however, make strong assumptions about information that will be made available to the learning system when new rules for a domain theory are required. Some earlier solutions to the problem of incompleteness have required encoding auxiliary domain theories (e.g., [Pazzani 88; Rajamoney 89]). Our contention is that such auxiliary theories are problematic for two reasons:

1. They place additional burdens on knowledge engineers, requiring that they not only encode a good domain theory, but that they encode a second theory that may be as difficult (or more difficult) to create than the first.
2. They are used under the same assumptions they are attempting to overcome. That is, the auxiliary theories are assumed to be complete, correct, and tractable.

Gemini makes use of additional knowledge in the form of contextual heuristics. However, these are domain-independent, are not expected to be complete, and can be automatically generated (by an algorithm described later) so that extensive engineering by a human is not required.

A second set of work addressing the problem of incompleteness has focused on developing interactive knowledge refinement systems that require the presence of an expert at all phases of learning (e.g., [Kodratoff 87] and [Wilkins 88]). While this is a valid approach, we have been motivated by the need for systems that can operate autonomously. Contextual heuristics make use of previously seen examples and explanations to automatically guide the creation of a new rule.

Yet a third class of solutions to the problem of incompleteness has resorted to using induction alone. Recall that incompleteness is detected when an explanation cannot be generated to show that a particular example is an instance of a certain goal concept. Solutions using induction alone often concentrate on learning a new definition for the goal concept that includes the training example in its class membership. Thus the domain theory retains classification accuracy for goal concepts, but loses some of the underlying structure of subgoals (e.g., [Cohen 91; Ourston and Mooney 90]). Our work concentrates on learning subgoals, or intermediate concepts, that may be useful in a variety of contexts. This is especially important for knowledge bases that are used to generate explanations for a user. Here the knowledge base must not only be empirically accurate, but must make sense conceptually.

Our contributions to explanation-based learning, with respect to the problem of domain theory incompleteness are threefold. We have developed a solution to the problem that avoids many restrictive assumptions of earlier approaches:

1. We do not require auxiliary domain theories that may suffer from the same problems as the original one. We make use of knowledge, but extract information that is readily available from the learning context.
2. We do not require the active presence of an expert either to provide missing rules or to tailor input examples. Examples for induction are selected, using contextual heuristics, from the system's history.
3. We learn definitions for the specific concepts that are required. We do not simply assure domain theory classification accuracy, but attempt to retain its underlying conceptual structure.

1.6.2. Contributions to SBL

Both a strength and a weakness of SBL is the fact it generally requires no explicit domain knowledge. When learning concept definitions as part of a larger knowledge base, however, there will often be information available that may be of use in guiding induction. Our solution to the problem of concept learning involves the encoding of heuristics that extract and apply contextual knowledge.

The use of explicit knowledge to guide induction is important for two reasons:

1. It allows the learning system to select positive and negative examples for induction, without requiring that an external source explicitly label examples as such.
2. It can help to generalize new definitions when large example bases are not available.

Our work shows that general contextual information is readily available and can be used to successfully guide induction. It minimizes the need for interaction with an external source since it allows for self-selection of instances for induction. It guides generalization when additional examples are not available. A major criticism of SBL systems is that they do not lend themselves to analysis of the influence that their components actually exert on induction. Our contextual heuristics are an explicit knowledge source. We provide an analysis of the influence that different types of contextual knowledge have on induction.

1.6.3. General Contributions to Machine Learning

We have developed Gemini to empirically demonstrate the utility of contextual heuristics for inductive learning of missing domain theory rules. More generally, we have used Gemini as a testbed for evaluating and comparing the influence of a set of the heuristics in four domains. Gemini is an important contribution to machine learning as it is one of the first examples of an integrated architecture successfully combining deductive and inductive techniques.

Though we concentrate on the problem of domain theory incompleteness in EBL, our technique applies more generally to any knowledge base of production rules in disjunctive normal form. One of our domains — the network fault domain introduced earlier — is in fact an expert system application.

1.7. A Summary of Results

The major results of our research may be summarized as follows:

- We have devised and implemented a solution to the problem of domain theory incompleteness. Our solution relaxes significant restrictive assumptions made by previous solutions. We do not require interaction with an expert, nor do we require an auxiliary domain theory. We make use of a domain-independent set of heuristics that automatically extract and apply contextual information.
- We show that the problem of incompleteness can be addressed as one of induction, but that information already available in the EBL system can be used to augment induction in a useful manner.
    - We show that induction of missing rules can be accomplished by considering examples already available to an EBL system, even if they are not pre-labeled as positive or negative for induction of a specific concept. As a result, an EBL system missing rules need not rely on an external source (human or otherwise) to provide such labels.
    - We show that learned rules can be generalized without waiting for additional examples, if contextual information is considered.
- We show that contextual heuristics that make use of explanatory information can be automatically generated by considering general attributes of EBL systems. They need not be devised by a system implementer.
- We identify a number of useful heuristics, obtained as a result of testing heuristics in four domains.
- We present Gemini, an integrated learning system that combines EBL and SBL for the mutual benefit of the two.

1.8. Reader's Guide

The following is a guide to reading the dissertation:

Chapter 2: Related Work - Here we survey previous work on the problems addressed. We begin with descriptions of explanation-based and similarity-based learning, and then follow that with discussions of work integrating the two to overcome the difficulty of incomplete domain theories. In addition, we briefly survey other work integrating deductive and inductive learning techniques.

Chapter 3: Gemini - Chapter 3 describes Gemini, our implementation of the methods we have devised to overcome the problem of domain theory incompleteness.

Chapter 4: Contextual Heuristics - Heuristics that make use of information extracted from partial as well as complete explanations are the foundation for our solution to the incomplete domain theory problem. Chapter 4 describes the origin of the heuristics, explains the intuition behind their derivation, and describes the algorithm we have devised to automatically generate the heuristics. The fact that our heuristics can be generated greatly simplifies the task of a knowledge engineer.

Chapter 5: Experiments - As indicated above, the information we apply to find rules missing from a domain theory are heuristics. Thus it is possible that they will fail. In Chapter 5 we describe experiments conducted that show that contextual heuristics are useful in learning rules needed by a domain theory. The experiments identify sets of heuristics that work most appropriately in each of the four domains we tested.

Chapter 6: Contributions and Evaluation - In Chapter 6 we discuss the contributions of our research. The primary contribution is that we have devised a solution to the problem of incompleteness that does not require the strong assumptions needed by earlier approaches. Chapter 6 evaluates Gemini with respect to closely related systems.

Chapter 7: Summary - Chapter 7 summarizes the dissertation and discusses limitations as well as future directions for our research.

Chapter 2

Related Work

2.1. Introduction

Much recent work on machine learning has grown out of expert systems research. During the development of early expert systems, it was learned that the task of knowledge acquisition is a formidable one. In order to facilitate this process a number of tools were developed.

There are two disparate machine learning approaches that bear on knowledge acquisition. These are explanation-based and similarity-based learning. As introduced in Chapter 1, the basic goal of an explanation-based learning system is to more efficiently recognize concepts that the system is already capable of recognizing. The learning process involves the analysis of an example of a concept in order to extract its characteristic features. Using a built-in domain theory (often a rule base), an explanation is derived describing why the instance is a member of the concept to be learned. The basic goal of a similarity-based learning system, on the other hand, is to acquire descriptions that will allow the system to recognize concepts it does not yet know. Many examples of a concept are given to the system. These are compared against each other in various ways to find a definition of the concept.

Both explanation-based and similarity-based learning methods have been applied with some success to problems in a variety of domains. Both methods have clear problems, however. Explanation-based learning assumes that a system will be supplied with an explicit theory of the domain of application that is complete, correct, and tractable. This assumption is clearly unrealistic for most complex, real-world domains. Similarity-based learning suffers because of its lack of an explicit theory of the domain. The method requires that human intervention play a large role in tailoring input examples and the language describing them in such a way as to allow a system to choose a set of features to define a concept. Less tailoring of the examples leaves a system open to the possibility of not converging on the best definition for a concept, or any at all, due to the computational complexity.

The primary goal of this research is to address the problem of incompleteness faced by explanation-based learning systems. A second goal is to introduce a domain-independent mechanism by which an inductive similarity-based learning system may *itself* focus on specific input examples and their features for learning.

The following section of this chapter describes explanation-based and similarity-based learning. Section 2.3 discusses the problem of incompleteness faced by EBL systems, describing previous work addressing this problem and contrasting it with our approach. (In Chapter 6 we revisit the most closely related systems, and do a more detailed comparison.) Section 2.4 discusses the use of deductive knowledge to guide induction, again with descriptions of previous work in the area. In both Sections 2.3 and 2.4 particular attention is given to systems that integrate explanation-based and similarity-based learning. Section 2.5 briefly introduces work on the problems of EBL and SBL that are not addressed in our research. We conclude with a summary of the chapter in Section 2.6.

2.2. An Overview of Explanation-Based and Similarity-Based Learning

2.2.1. Explanation-Based Learning

Explanation-based learning (EBL)[2] (e.g., [Ellman 89a; DeJong 81; DeJong 83; DeJong and Mooney 86; Mitchell et al. 86; Silver 86; Winston et al. 83]) is a deductive machine learning approach in which a definition of a concept is derived, usually after observing only a single example of that concept. To understand this method on an intuitive level, consider a robot operating in a room containing household objects. In order to manipulate different objects in its world, the robot must have a mechanism for recognizing them. For example, it might require the ability to recognize cups.

---

[2]EBL is sometimes referred to in the machine learning literature as explanation-based generalization (EBG).

Consider a robot that has been provided with a knowledge base of rules describing objects in its world and their characteristics. The rule base would be the domain theory as it constitutes the robot's representation of its domain of application. Some rules from this domain theory might be:

```
LIGHT-WEIGHT(X) ∧ PART-OF(X,Y) ∧ ISA(Y,HANDLE)
    ⇒ LIFTABLE(X)
```

that states that if an object is light and has a handle then it is liftable, and:

```
ISA(X,OPEN-VESSEL) ∧ STABLE(X) ∧ LIFTABLE(X)
    ⇒ ISA(X,CUP)
```

that states that if an object is an open vessel that is liftable and stable then it is a cup.[3] Now assume that the objects in the world are represented by predicates that describe basic structural features of the objects, such as FLAT and LIGHT-WEIGHT. Because the domain theory relates these primitive predicates to higher level concepts such as CUP, it can be used by the robot to deduce that a particular object is or is not a cup. The deductive process, generally implemented by a theorem-prover, is computationally expensive, however. A major goal of explanation-based learning is to improve the performance of such systems.

Input to EBL is the name of the concept to be learned, an instance of it, and a domain theory to be used to prove that the input instance is an example of the given concept. In the robot domain described above, the concept to be learned might be ISA(X,CUP). The example of a cup might be the representation of a particular blue ceramic cup: a conjunction of primitives such as COLOR(obj1,BLUE), MADE-OF(obj1,CERAMIC), etc. The domain theory would include rules such as those given above.

The goal of EBL is to create a general concept description based upon the

---

[3]These examples are modified from Mitchell *et al.*'s paper on a unifying framework for the method of explanation-based learning [Mitchell et al. 86]. All examples from this domain presented here are based upon examples from their paper.

predicates describing the particular example, such that the general description is justified by the proof derived to show that the example was a member of the concept to be learned. The first step of the learning process is to use the domain theory to generate such a proof. The proof is sometimes called an explanation. The explanation for the input described in the preceding paragraph is shown in Figure 5.[4] Note that not all predicates used to describe the input example are actually found in the explanation. It is assumed that these, for instance the color of obj1, are not relevant to the definition of a cup. Though there may be many ways to explain that an input example is a cup, a specific single explanation is generated for the *given* instance.

Figure 5: Explanation of Cup

The next step of EBL is to maximally generalize the description of the input example such that the derived proof structure still holds. Many constants appearing in the proof can be made variables as shown in Figure 6. In some cases the generalization must be constrained. For example, OPEN-VESSEL was not generalized. The general description of a cup is shown in Figure 7. It is obtained by extracting references to observable features from the explanation. It is correct because it is validated by the deductions of the explanation from which it was taken. It can be used by the robot to

[4] Modified from [Mitchell et al. 86], page 59.

efficiently recognize cups, because it can now simply apply a single rule rather than having to construct a proof. (It has more recently been shown by [Minton 88] that there are cases in which the application of a single rule is less efficient than proof construction due to the complexity of determining that the rule applies.) In domains such as that of solving algebraic equations, where proofs can resemble traces of problem solving steps, the generalized proofs may be stored to be applied automatically to analogous examples later. The result of explanation-based learning as described here is not to have learned a concept that could not have been recognized, but to learn a more efficient definition of such a concept.

Figure 6: Generalized Explanation of Cup

The discussion above is intended to give an intuitive view of explanation-based learning. A more formal discussion is beyond the scope of this chapter. (But see [DeJong and Mooney 86; Mitchell et al. 86]; a survey of EBL systems is presented in [Ellman 89a].) The key ideas of EBL are:

- Explanation-based learning is a method in which a single example is analyzed and then generalized by a system.
- In order to perform the analysis, a system must possess a domain theory that is complete and correct, and the process of analysis must be tractable.

Figure 7: General Definition of Cup

- The analysis, or derivation of an explanation, will often be performed by a deductive inference mechanism such as a theorem-prover. Depending upon the domain, an explanation may be a deduction of the flavor in the example above, or, more generally, any transformation of an initial problem state to a goal state.
- Generalized explanations, in addition to learned general definitions, may be saved so that they need not be re-derived.

A summary of EBL is given in Figure 8.

```
INPUT:   name of a concept to be learned;
         specific example of the concept;
         domain theory.

OUTPUT:  more general (efficient) definition of the concept;
         generalized explanation.

METHOD:  1.  derive a proof that the example is an instance of
             the concept.
         2.  maximally generalize the proof while maintaining
             its correctness.
         3.  extract a general definition from the generalized
             explanation.
```

Figure 8: Summary of Explanation-Based Learning

2.2.2. Similarity-Based Learning

Similarity-based learning (SBL) (e.g., [Fisher 87; Lebowitz 87; Michalski and Stepp 83; Mitchell 78; Quinlan 86; Winston 72]) is an empirical technique that involves comparisons among potentially large numbers of input examples. The input examples are compared in order to find similarities and differences among them. Similarities are generally assumed to define a useful concept.

To get a flavor of the method, consider the robot world described in the preceding section. Our same robot is operating in a room containing household objects and must be able to recognize cups. Unlike in the EBL setting, this robot does not have a domain theory relating predicates describing basic structural properties of the objects to higher level concepts. It must learn the descriptions of the objects ‘‘from scratch’’.

Figure 9: Sample Input to SBL

Input to the learning algorithm is the name of the concept to be learned, for instance CUP, and examples of that concept. Additional input to the algorithm might be examples of objects that are not members of the concept. Sample input examples to an SBL system that is to learn the concept CUP are shown in Figure 9. Structural properties that are similar to those used in the preceding section are used to describe the inputs here. A goal of SBL is to create a general description of the concept that, to the maximal extent, includes all the positive examples and excludes all the negative ones.[5]

---

[5]There may be additional criteria: parsimony, for instance.

The most obvious algorithm for doing so is to compare the input and to find the most specific set of descriptive features shared by all of the positive examples but not by the negative examples. A general description for the CUP concept based upon the input shown in Figure 9 is given in Figure 10. Note that though the feature value of ceramic was shared by the negative example, the remaining features constrain the definition of CUP so that it excludes this example.

Figure 10: General Definition of Cup Found by SBL

The intuition behind the method is that the features essential for defining "CUP-ness" must appear in all examples of cups. The features that are not relevant to defining the concept, such as color, will vary among the examples and thus not be included in the general description. The version of SBL presented here is a simple-minded one. Many SBL systems deal with more complex representations or disjunctive definitions, for example.

Though they do not possess the explicit domain theories of EBL systems, SBL programs do have an implicit bias built into them that allows them to arrive at particular general concept definitions [Mitchell 80; Utgoff 86]. For instance, in generating the CUP definition of Figure 10, the algorithm did not recognize the similarity in the names of the CUP owners. The capacity to look for such similarities was simply not encoded in the learning algorithm. The implicit bias of SBL systems includes, among other things:

- the language in which input examples and the learned concepts are represented.
- concept hierarchies that define relationships among the values that can be taken on by features.

Bias built into SBL systems is called inductive bias, because the generalization performed by such systems is analogous to that of inductive reasoning.

The example of similarity-based learning presented here belongs to just one variety of the methods called SBL. Although these methods vary, they share the intuition that the discovery of commonalities among members of a class may lead to a description for the class that will have predictive power in analyzing future examples. Most generally, SBL systems may be divided into two categories based upon whether or not examples provided to the system have been classified by an external source.[6]

Those methods that receive input classified by an external source are termed **learning from examples**. Work in this area includes that of [Mitchell 78; Quinlan 86; Winston 72] among others. These systems in turn may be divided on the basis of the types of examples input. Some are given both positive and negative examples of a concept, while others receive only positive examples. Another distinction may be made on the basis of whether all examples are seen simultaneously or whether a description is built incrementally by considering one example at a time.

Those methods that receive unclassified input are referred to as **conceptual clustering**. Typically, many examples of a number of concepts are presented to a conceptual clustering system simultaneously. The system looks for subsets of the input examples such that the elements of each subset share a number of descriptive features not shared by the members of any other set. Work in this area includes that of [Fisher 87; Lebowitz 87; Michalski and Stepp 83].

As with the overview of explanation-based learning above, this chapter does not present a formal treatment of similarity-based learning. A summary of the learning from examples style of SBL is given in Figure 11.

[6]This description of SBL has been modified from a taxonomy of learning methods given in [Machine Learning, Volume I 83].

```
INPUT:   name of the concept to be learned;
         specific examples (positive and/or negative)
           of the concept.

OUTPUT:  general definition of the concept.

METHOD:  compare examples to find similarities among positive
         examples and differences of positive examples from
         negative.
```

Figure 11: Summary of Learning from Examples

2.2.3. A Comparison of Explanation-Based and Similarity-Based Learning

Explanation-based and similarity-based learning methods are approaches that can be placed on opposite ends of a spectrum describing purely deductive to purely inductive techniques [Mitchell 84]. In explanation-based learning, analysis of a single example guides the generalization of knowledge possessed by a system in order to make itself more efficient. Its power derives from having extensive domain knowledge. The dependence upon the domain theory is also a weakness of the method. The theory is assumed to be complete and correct and the explanation derivation process to be tractable. These assumptions are unrealistic in many complex, real-world domains. Similarity-based learning systems do not possess the extensive, explicit knowledge bases of EBL. They must instead have extensive explicit example bases which guide them in a search through all possible concepts representable in a given language. Problems may arise if the inductive bias is not sufficiently strong to guide the search or if the input data is noisy. A more fundamental problem for both learning methods is the possibility that the concept to be learned cannot be adequately represented in the language of the system.

A natural solution to the problems of incomplete, incorrect, and intractable domain theories for EBL is to augment such systems with SBL, which works in the absence of a strong domain theory. Similarly, the deductive reasoning from an explicit domain theory of EBL may be added to SBL systems to prune the search space for interesting similarities or to transform one representation language into another. The following sections describe in more detail the two specific problems of EBL and SBL addressed in our research. Section 2.3 contains case studies of previous work on the problem of incompleteness for EBL, with particular attention given to approaches that involve the integration of explanation-based and similarity-based methods. Section 2.4 similarly describes work on the problem of providing focus for similarity-based learning. Examples illustrating each case study are taken from the domain to which it has been applied.

2.3. The Problem of Incomplete Theories for Explanation-Based Learning

Rajamoney and DeJong [Rajamoney and DeJong 87] identify two ways in which domain theories used by EBL systems can lack knowledge.[7] In the first, an explanation, or proof, cannot be completed by the EBL system because knowledge, usually in the form of a rule, is missing from the domain theory as illustrated in Figure 12. In order to complete the explanation, a rule or fact must be added to the domain theory. In the second case, proofs can be constructed leading to a conclusion, but they lack the detail required for a particular application. This problem may be seen as one of an incorrect choice of granularity of the knowledge represented. Here Rajamoney and DeJong's second description of incompleteness is treated as incorrectness of the domain theory. This section discusses five approaches to the problem of missing rules.

Figure 12: The Problem of Incompleteness

[7] In defining incompleteness types, incorrectness and intractability are ignored.

2.3.1. Rajamoney

Rajamoney [Rajamoney et al. 85; Rajamoney 88; Rajamoney 89] proposes a method called **experimentation-based theory revision**, implemented in the ADEPT system, as a solution to the problem of incompleteness in a domain theory. Michalski and Ko [Michalski and Ko 88] have also discussed the use of experimentation in addressing this problem. The discussion here focuses on Rajamoney's work.

ADEPT's domain theory is represented using Forbus's Qualitative Process Theory [Forbus 84]. According to this theory, changes in the world such as boiling or evaporation are due to *processes,* which in reasoning may be used analogously to rules. Processes are composed of three parts:

1. individuals - a set of objects that participate in the process,
2. preconditions and quantity conditions - a set of conditions that must be satisfied if the process is to be active,
3. relations and influences - a set of statements about the world that are true if a process is active.

Preconditions and quantity conditions are analogous to the *if* part of an *if-then* rule, while relations and influences are analogous to the *then* part. ADEPT's goal is to use processes in explaining viewed changes in the world.

Rajamoney presents three ways in which incompleteness can become evident in a domain theory of processes:

1. an active process, i.e., one whose preconditions are satisfied, has flawed influences and is, in fact, causing the phenomenon that ADEPT is trying to explain. Essentially, this means that the phenomenon should be in the *then* part of an existing active rule (or should be deducible from a chain of active rules), but is not.
2. an inactive process that could explain the phenomenon if active has flawed preconditions or quantity conditions and should be active. In other words, a rule in which the phenomenon appears in the *then* part should have fired, but did not. Its preconditions, or *if* part, must be modified so that the rule may become active.
3. a new process is causing the phenomenon. No modification of an existing rule is sufficient to complete the explanation.

ADEPT proposes new rules when it is unable to complete an explanation for a viewed change in the world. The types of rules proposed reflect only the first two of the descriptions of inadequacy just discussed. That is, an existing active rule can be modified to include the viewed change in its *then* part, or an existing inactive rule can be modified so that its preconditions are satisfied. These constrain the number of rules that are proposed. Proposed rules are empirically tested for validity. The testing performed by ADEPT is not SBL per se, but rather empirical validation. It is included here, however, because the underlying premise of SBL is empirical validation. That is, SBL makes the assumption that patterns observed over many examples will continue to be observed. ADEPT assumes that if the proposed rules accurately explain other observed changes in the world then they will continue to apply.

In order to minimize dependence upon a user, ADEPT tests its proposed rules by designing experiments so that the results of some of them will be inconsistent with a number of the proposed rules. These rules are eliminated from consideration. In an example taken from [Michalski and Ko 88], a system is given the task of explaining why a wine bottle placed in a freezer shattered. In the absence of more specific information, the system might propose that cold causes glass to contract while the volume of the contents of the glass remains the same. Alternately the liquid might expand. An experiment could be devised in which a glass container of water was placed in a freezer. If the container did not break, the first proposed rule would be invalidated.

Presumably ADEPT must possess a theory of experiment design for the domain to which it is applied. Given that good experiment design is a non-trivial problem, such a theory must be complex and would therefore be subject to the problem it is supposed to address, that of incompleteness. In contrast, Gemini, as described in Chapter 3, employs a knowledge base of heuristics that make use of explanations (both partial and complete) in guiding induction. As the heuristics in Gemini are used only as a guide, they are not required - or expected - to be complete.

A positive aspect of Rajamoney’s work is that the number of newly proposed rules is constrained. This is important because it would be computationally infeasible to test all possible rules for validity. It is not clear, however, that the rule proposition mechanism is constrained sufficiently to overcome the problem of time-intensive experimental (in)validation. In addition, because the rules to be tested are constrained, a large class of potentially missing rules (or processes) may never be found. In particular, ADEPT does not handle cases in which both a premise and conclusion must be changed. Gemini is able to propose rules that are not mere modifications of existing ones. In focusing on modifications of existing rules, ADEPT assumes that the existing domain theory is nearly perfect. Though reasonable in many cases, this assumption may be too optimistic in others.

Finally, ADEPT is limited to applications where experimental validation can potentially be performed.

2.3.2. Hall

Hall describes a method for learning new rules called **Learning by Failing to Explain** [Hall 86]. In this method, a system that fails to find an explanation for an input example is given a new, analogous example by a teacher. The system learns by analyzing the analogue and comparing its explanation using similarity-based methods to the incomplete explanation.

Hall's system works in the domain of logic circuit design. Given as input a function name and an implementation of that function as a logic circuit, the system's goal is to prove that the implementation given is correct. Rules in the domain theory of the system have the form LHS ⇒ RHS, where LHS denotes a function, such as *PLUS*, and RHS is the description of an implementation for the LHS. The RHS may refer to other functions as well as to specific circuit implementations. For example, as shown in Figure 13a, a function F0 might be decomposed into three subfunctions F1, F2, and F3, which might be implemented by circuits x, y, and z, respectively. The tree given in the figure is taken to be a proof that the implementation is correct. Rules used to construct the proof might be:

```
F0 ⇒ F1 F2 F3
F1 ⇒ x | q | r | s , where | indicates disjunction
F2 ⇒ y
F3 ⇒ z
```

Figure 13: Proof Trees for Logic Circuits

In Hall's system an explanation failure arises when the system is unable to prove that an input structure implements a given function. In the above example, this would occur if the system were missing the rule F1 $\Rightarrow$ x | q | r | s as depicted in Figure 13b. When the system signals a failure, a teacher provides an example from which the system can learn the rule it needs. It is assumed that in the failed proof the system is unable to link at most one subfunction to structures within the given implementation. That is, at most one rule is missing from the proof, and it directly links the unexplained subfunction to features of the implementation.

The teacher gives the system a new input structure that:

1. implements the same function as that given to the system initially and
2. contains as a subfunction the one for which the proof could not be completed.

A teacher-provided analogue to the example given above might be one for which a proof is given in Figure 13c. The system derives as complete an explanation as possible for the new input. Assuming that all rules linking a specific circuit implementation to subfunction F1 are missing, a partial proof for the analogue would look like that in Figure 13d. Next, corresponding parts of the two implementations are matched, where matching is defined as functional equivalence - in essence, the proofs are matched. In this example, suppose the system determined that F2 and F4 are *functionally* equivalent, as well as F3 and F5. The system assumes that corresponding unmatched parts are equivalent to each other, i.e., that they both implement the same function. In the example above, a and y match since F2 and F4 are functionally equivalent; similarly, b and z match. The yet unmatched parts, q and x, are assumed to both implement the unproven function F1. Rules corresponding to the separate implementations are created and generalized. For the example above, the system would learn the rule: $F1 \Rightarrow x \mid q$.

Hall’s system, like Rajamoney’s, works under the optimistic assumption that its domain theory will be very nearly complete. This must be the case if at most one subfunction may be unproven in the explanation of an entailing function. In contrast, Gemini allows for the possibility of missing multiple rules in a single explanation. Hall’s method is dependent upon a user not only for the nearly complete theory, but also for training examples from which to learn, which must be specially tailored to the failure situation. Gemini was specifically designed to minimize the need for a teacher, assuming that a system might be required to operate autonomously. Fortunately the input provided by the teacher in Hall’s method does place strong constraints on the number of new rules proposed by the system. Recall that in Rajamoney’s work the number of new rules proposed, although somewhat constrained, could be potentially large.

A final constraint in Hall’s work is the choice of domain. His method provides focus on relevant parts of a missing functional implementation by eliminating from consideration all parts already accounted for. This may fail in other domains. Even in his domain, a single part of an implementation may play dual roles. (Though one heuristic in Gemini mimics Hall’s assumption, it need not be applied in all domains. If one was not confident in the heuristic, one could choose to apply other heuristics.) Hall’s method also requires that functional equivalence be determined to detract attention from those implementation features already accounted for. In another domain, this might require that, for instance, an analogy be found, which is a large problem in itself.

2.3.3. Pazzani

Pazzani, in his system OCCAM [Pazzani et al. 86; Pazzani 87; Pazzani et al. 87; Pazzani 88], which predicts and explains the outcome of events, uses general knowledge about causality to propose cause/effect rules to be added to an incomplete domain theory. Pazzani states a clear preference for knowledge-based methods over empirical ones. He believes that EBL is always to be preferred to SBL and that a system should only fall back on SBL as a last resort. In addition to the domain theory used by EBL, he provides OCCAM with a base of *generalization rules* that function as templates for new rules to be created.

Input to OCCAM is a description of an event. OCCAM's goal is to predict the outcome of it. The chain of reasoning from the description of the event to the predicted outcome is a proof that the outcome will occur. A failure occurs if a proof cannot be derived that links aspects of the event to a predicted result. When this happens, OCCAM instantiates a generalization rule that will complete an explanation. Generalization rules encode information reflecting a theory of causality, but are otherwise independent of a domain. In theory they could be used by a performance system that predicted the outcome of meteorological events as well as by another that predicted the outcome of chemistry experiments. A typical example of a generalization rule is: ''If an action on an object precedes a state change for the object, then the action may cause the state change.'' Given two examples, one in which a balloon could not be blown up and one in which it was first stretched and then blown up, OCCAM would propose a rule that stated that stretching a balloon causes it to be in a state from which it can be blown up. Rules are tentatively proposed by OCCAM, as the generalization rules are not guaranteed to be instantiated correctly in all cases. For example, if a balloon were placed in water before being blown up, OCCAM would propose a rule that stated that dipping the balloon in water caused it to be blown up later. Rules are validated empirically as more examples are seen by the system. Thus, at any time, OCCAM's rule base may be incorrect.

A major strength of OCCAM is that the rules proposed by the system are constrained by the base of generalization rules which are applicable across a number of domains. However, they are specific to domains involving causality. It is not clear that an analogous base of rule templates could be devised for, say, Hall's logic circuit domain. Nor would such templates carry over to the classification domains to which Gemini has been applied. Unlike Hall's system, however, Pazzani's is not dependent upon the presence of a teacher. OCCAM proposes a tentative rule based on a single example and then waits for later examples to confirm or deny the rule. Unlike Gemini, however, it is not able to extract other potentially relevant information from its previous experience.

2.3.4. Kodratoff and Tecuci

DISCIPLE [Kodratoff and Tecuci 87; Kodratoff 87] uses an incomplete domain theory to pose questions to a user who then teaches the system missing rules. DISCIPLE has been applied to the domain of designing technologies for the manufacture of loudspeakers. A typical explanation is a plan for the manufacture of a loudspeaker. When the system is unable to complete a plan because knowledge is missing from the domain theory, a user supplies a rule that will complete the plan. DISCIPLE uses a combination of explanation-based and similarity-based learning to generalize that rule, thereby making it more widely applicable.

DISCIPLE begins the process of rule generalization in explanation-based mode, trying to explain why the rule given by the user is valid. Suppose that during the phase of constructing a plan, DISCIPLE had needed to *ATTACH sectors ON chassis-membrane-assembly*. Suppose that a user provided the following as a method to achieve that goal: *APPLY mowicoll*[8] *ON sectors, PRESS sectors ON chassis-membrane-assembly*. It is not assumed that the domain theory can provide a complete explanation, but only that it contains enough information to propose partial explanations. A best partial explanation is selected by the teacher. In the example described above, a user selected the explanation in Figure 14a as the best explanation proposed by DISCIPLE. Next DISCIPLE enters an SBL phase in order to generalize the rule. It searches the domain theory for information that matches the graph edges of the explanation selected. For example, the partial explanation in Figure 14b was extracted

[8]From [Kodratoff 87], page 271. *Mowicoll* is roughly translated as *Romanian glue*.

from DISCIPLE's domain theory as being similar to the first partial explanation. Before the matching and generalization of SBL can proceed, the user must validate all examples as being similar and relevant in the current context.

Figure 14: Partial Explanations in DISCIPLE

Kodratoff and Tecuci's system relies heavily upon interaction with a teacher, who gives the system the information it is missing. Rather than counting on the teacher to provide a general rule, however, it asks for one appropriate for a particular situation and uses explanation-based and similarity-based techniques to guide the teacher in providing it with information sufficient for generalization of the rule. Constant supervision by the teacher can stop DISCIPLE from choosing irrelevant examples and assures correctness of the learned rules. This, however, places a heavy burden on the teacher. More recent work on the CLINT system [De Raedt and Bruynooghe 89] is similar to that presented here on DISCIPLE.

2.3.5. Wilkins

ODYSSEUS, developed by Wilkins [Wilkins 88], is an apprenticeship learning system that improves an expert system by watching an expert. ODYSSEUS is designed to improve a knowledge base of the type used by the HERACLES expert system shell. HERACLES is essentially the NEOMYCIN expert system with the medical domain knowledge removed [Clancey 84]. Experiments run on ODYSSEUS were performed using the NEOMYCIN medical knowledge base for diagnosis of neurological problems.

HERACLES uses *heuristic classification* to solve problems. This is a process in which a solution is chosen from a pre-specified set using heuristics. Three distinct types of knowledge are used by HERACLES: domain knowledge represented as MYCIN-like rules [Buchanan and Shortliffe 84]; problem state knowledge that indicates information such as the rules that have been applied during a diagnostic session; and strategy knowledge that is an explicit representation of various types of classification heuristics. A strategy metarule might state that one can find out about a symptom by finding out about a symptom that subsumes it.

During a diagnostic session, an expert requests specific data on a patient in order to form a diagnosis. Using its strategy metarules and domain knowledge, ODYSSEUS generates an explanation indicating why a question asked by the physician is logical given the path being followed to the final diagnosis. An explanation is created for each problem-solving action. During the course of operation, ODYSSEUS may fail to generate an explanation. If a proof fails, it is assumed that a piece of either domain knowledge or problem state knowledge is missing. ODYSSEUS does not learn new strategy knowledge.

Say that in diagnosing a case of acute meningitis, a physician has requested information about the presence of photophobia - and that photophobia is present. If ODYSSEUS does not find in its knowledge base a link indicating that photophobia is evidence for acute meningitis, it could propose this as a rule. Proposed rules are confirmed using a separate induction system. This is more complicated than is apparent at first glance due to the varied nature of the domain rules. Many are of the type that indicate a symptom as evidence for a specific problem. Others provide clarification information. For instance, information about the duration of a headache describes the headache in more detail, but need not alone indicate anything specific. It may eliminate potential diagnoses, but is not necessarily enough to focus on any single one. In all, there are thirty-three different types of rules according to Wilkins. He has implemented strategies for finding three of these types.

Although Wilkins has implemented strategies for determining only three of the rule types, his work does provide coverage over a greater range of rule semantics than does the other work presented in this section. One of the strengths of ODYSSEUS is that full diagnostic traces by experts provide the system with the fine granularity that allows it to focus on the information missing from the knowledge base. The problem with this, however, is that the diagnostic traces must be available to the system. Furthermore, they must be correct, and must follow the same strategies encoded in the system's strategy metarules. Finally, the base of strategy metarules is assumed to be correct and complete. As with Rajamoney's theory of experimentation, this may be unrealistic to assume. Even if complete, it provides coverage only for diagnostic domains, as Pazzani's does only for causal domains.

2.3.6. Summary

In this section we have described five approaches to the problem of finding new rules for an incomplete domain theory. Wilkins, as well as Kodratoff and Tecuci, take a conservative approach, which relies very heavily on a user for essentially providing the information to be included in a new rule. Hall also relies on a user to input an analogue to a failure situation from which the system can learn. His system is left to analyze the analogue as well as to generate a new rule from it. Although not discussed here, [Laird et al. 90] and [Genest et al. 90] similarly rely on user guidance to provide the information to be included in a missing rule. Our work is based on the observation that such detailed guidance is not always available to a system. A goal of our research is to minimize the interaction required with an expert during the course of learning. Gemini, described in Chapter 3, assumes that it can only use information internal to it, either in the form of built-in knowledge or in the form of processing history.

Both Pazzani's OCCAM and Rajamoney's ADEPT are independent of a teacher during the learning process, but they rely on the existence of auxiliary theories to guide them in creating new rules. ADEPT needs a theory of experimentation, while OCCAM has one of causality. Our contention is that these solutions simply push the problem to a different level, as the auxiliary theories cannot be assumed to be complete. In contrast, our system Gemini, while using heuristics to learn rules, uses heuristics that are domain independent. They rely primarily on the structure of explanations, rather than on their semantics. Furthermore, they are used only as a guide, and are therefore not expected to be complete.

In general, the systems discussed in this section focus attention when learning a new rule either by receiving assistance from a teacher, or by relying on an auxiliary theory. Gemini eliminates the need for interaction with a human while relaxing constraints on the quality required of an auxiliary theory. Chapter 6 compares Gemini to related methods in more detail.

2.4. The Problem of Providing Focus for Similarity-Based Learning

Similarity-based learning may be viewed as search through the hypothesis space of representable concepts for one that is consistent with all positive input examples and inconsistent with negative input examples. The hypothesis space is potentially huge. A problem to be addressed is how search may be directed to prune large parts of the hypothesis space.

[Mitchell 84] was among the first to propose that explanation-based learning be used to focus the attention of SBL, enabling it to make larger leaps through its hypothesis space of concept definitions. Mitchell's proposed solution is an extension of the version space algorithm [Mitchell 78] for SBL. A version space is a lattice of representable concepts. Representations of specific instances are at one end of the lattice, while the most general concept definition, of which all entities are members, is at the other end. The version space algorithm maintains two sets of concepts. One set, S, contains the most specific concept definitions consistent with all the positive examples seen by the system. The other, G, contains the most general definitions not inconsistent with any of the negative examples seen. The algorithm learns incrementally, i.e., concept definitions are modified after seeing each input. Positive input examples may cause the S set to be modified by making its members more general. Negative input examples cause the G set to be modified, making it more specific so as to exclude the negative examples. S and G sets are pictured in Figure 15. After seeing enough input, the S and G sets should grow toward each other and become equivalent. This is the learned concept definition.

Figure 15: Boundary Sets of the Version Space

Mitchell suggested that performing EBL on each positive input example provides a set of sufficient conditions for being an instance of the concept to be learned. This set of sufficient conditions provides a basis for refining the S set of the version space. Negative instances are still used to refine the G set. This has recently been implemented by Hirsh [Hirsh 89].

A number of researchers have more recently investigated the idea of using EBL to provide extra information to SBL in order that it might intelligently prune its search space. The following sections present the approaches of Salzberg, Swaminathan, and Van Lehn.

2.4.1. Salzberg

Salzberg, in HANDICAPPER [Salzberg 83; Salzberg 85], derives explanations to identify potentially important features for revising the definition of a concept. HANDICAPPER's domain of application is horse racing, where its performance task is to predict the horse that will win a race. The learning task is to identify sets of features that define the concepts ‘‘winning horse’’ and ‘‘losing horse’’. Input to HANDICAPPER are feature sets describing the horses in a race. HANDICAPPER chooses a winner by selecting the horse with features most similar to those describing a winner in general. If the prediction is incorrect, then the concept definitions for winning and losing horses are modified.

To modify the feature sets defining the two concepts, HANDICAPPER performs SBL, looking for *differences* between the horse predicted to win and the horse that actually won. This provides the system with a choice of features to be modified in the concept definitions for winning and losing horses respectively. Next explanations are derived that enable the system to identify those features most likely to require modification. In HANDICAPPER explanations relate features of a single horse to each other. The domain knowledge contains rules linking the presence of certain features to others. Concentrating on those features found to be different in the SBL phase, the EBL phase looks for those that contradict rules in the domain knowledge. No generalization of the explanation is done. The features identified by the EBL phase are modified in the concept description.

Handicapper makes use of a knowledge base of horse racing information to guide the induction of definitions for winning and losing horses. Ultimately, the definitions are not permitted to contradict the domain theory. This is a reasonable way to restrict the generalization of a concept definition, but it assumes that the domain theory is perfect. Even if this could be guaranteed in the horse racing domain, it would be unlikely to hold in other domains. In Gemini, on the other hand, induction is performed for the purpose of completing an imperfect domain theory.

Induction in Handicapper occurs incrementally. A problem with incremental learning is one generally also found in hill-climbing systems. The learned definition may approach a ‘‘local best’’ which might not be the most appropriate definition overall. Gemini performs incremental learning so that it need not wait to see all examples. However, it retains examples so that a concept definition may be revised to reflect a more global perspective later.

2.4.2. Swaminathan

Swaminathan, whose system has been applied to the domain of epidemiology [Swaminathan 88], uses similarity-based learning to identify a set of features common to the victims of an illness in a manner similar to Salzberg’s for horse racing. Because the feature set describing victims of an illness can potentially be huge, Swaminathan uses EBL to identify a set of features on which to focus. The domain theory used by the explanation-based component of Swaminathan’s system is a partial causal theory of disease. The parts of the domain theory that are missing are those that refer to specific values of features describing input examples. For instance, given a set of hepatitis victims for which the system needs to find the cause of the illness, the domain theory would contain general information about contaminated water sources possibly causing hepatitis. It would not contain more specific information stating what water sources were contaminated. Explanations are generated using the partial theory linking general features of the input examples to the illness. As an explanation does not refer to specific feature values, it is not a proof that those features referred to are *actual* causes. It is a proof that those features are *possible* causes. SBL concentrates on these, presumably relevant, attributes of the inputs when searching for the attributes shared by the positive examples (the victims of a disease).

Like Salzberg, Swaminathan assumes that the domain theory will provide appropriate focus on the features relevant to a concept definition. As in Gemini, Swaminathan relaxes the constraint that the theory be complete. Gemini takes this a step further, however, by encoding a knowledge base of general heuristics that can make use of the explanatory knowledge in a number of different ways. The knowledge can be used to focus on example features not only due to their presence in an explanation, but due to the configuration of the explanation. Gemini’s heuristics allow it to identify negative examples of a concept to further aid the induction. Swaminathan’s system makes no use of negative examples.

2.4.3. Van Lehn

In Van Lehn's SIERRA system [Van Lehn 87], guidance as to the relative importance of features and the way in which to generalize them using SBL is provided by a teacher. Certain conventions, or **felicity conditions**, are understood by both the similarity-based learning system and the teacher. For example, in SIERRA's performance domain, arithmetic procedures, it is understood that the teacher will introduce at most one new disjunct to a concept per lesson. This drastically cuts the hypothesis space to be searched by SIERRA in performing inductive inference. Because of its dependence upon a teacher and felicity conditions between the teacher and learning mechanism, this work is of a somewhat different nature than the systems above. However, it is included here because of an explanatory step that SIERRA performs before doing induction. Upon receiving an input example in the form of a sequence of actions that solve an arithmetic problem, SIERRA attempts to parse the action sequence. The parse tree constitutes an explanation for the action sequence. The inability to complete a parse indicates the new disjunct, in SIERRA's case a subprocedure, to be learned. Due to the nature of the learned disjunct - i.e., that it is a new arithmetic rule - this may also be viewed as a use of induction for the incomplete theory problem of EBL.

2.4.4. Summary

Using knowledge to guide the otherwise slow and inexact process of induction is a tempting idea. This section discussed four approaches to cutting the search of SBL. Mitchell proposed that a positive example generalized by EBL be input to the version space algorithm. The single generalized example is essentially equivalent to a set of ungeneralized positive examples. This approach is problematic in that it assumes completeness of the domain theory used by EBL. Salzberg and Swaminathan use explanations to focus the attention of SBL on features that are presumably most relevant. This approach does not necessarily require that explanations be complete. Partial explanations will often simply provide less information. In both cases, however, the domain theory is treated essentially as if it were perfect. In contrast, Gemini has an additional knowledge base of heuristics that suggest a variety of ways in which explanatory information might be used as a guide. Though it is generally assumed that they will work well, none of the heuristics are assumed to work perfectly in all cases. Additionally, Gemini is able to use its heuristics to identify negative examples that might not explicitly have been available for learning a particular concept. Mitchell's approach and Salzberg's make use of negative examples, but they must be explicitly input by the environment. Swaminathan does not make use of negative examples at all.

Van Lehn relies on felicity conditions between a teacher and learning system in order to focus incremental inductive learning. As our goal is to minimize the need for interaction with an expert, or teacher, such an approach is inadequate for our purposes.

2.5. Other Related Work

Due to their complementary aspects, integrating explanation-based and similarity-based learning is a natural solution to overcoming problems in the individual methods. In Sections 2.3 and 2.4 we discussed previous applications of such methods to the problems addressed in our research: incompleteness of explanation-based learning domain theories and focus for similarity-based learning. In the following sections we briefly describe the integration of EBL and SBL to overcome problems not addressed by our research. We also mention additional work in the integration of deduction and induction.

2.5.1. Integrating EBL and SBL to Overcome other Deficiencies of the Methods

2.5.1.1. The Problem of Incorrect Theories for Explanation-Based Learning

One manifestation of the incorrect theory problem of explanation-based learning was introduced in Section 2.3 on incompleteness. Incorrectness may be the result of insufficient detail in the domain theory. This occurs when the designer of the theory makes assumptions about knowledge that may be approximated. The use of such a theory may result in mutually inconsistent explanations as depicted in Figure 16. A second type of incorrectness, probably the more basic one, is that in which invalid knowledge exists in the theory and must be retracted.

The first manifestation of incorrectness is addressed using SBL techniques by

Figure 16: The Problem of Incorrectness

[Araya 84], [Silver 88], and [Bergadano, et al. 87]. [Smith et al. 85] also address the problem, but with the aid of a detailed theory underlying the abstract one that is actually used. [Ginsberg 88] solves the problem by doing a massive initial check for redundancy and inconsistency. The second manifestation of incorrectness is addressed by [Dietterich and Flann 88].

2.5.1.2. The Problem of Intractable Theories for Explanation-Based Learning

A problem arises in explanation-based learning when the deductions that the system might make are computationally prohibitive and cannot be completed [Mitchell et al. 86]. While constructing an explanation the system exceeds the resources (time, memory, etc.) allotted to it. This is called the problem of intractability of the domain theory.

The knowledge represented in the domain theory of an EBL system may be viewed as a collection of *if-then* rules. The process of constructing the proof that the input example is a member of the concept to be learned is one of chaining together these rules such that the description of the example is linked to the concept. Given this, two different kinds of intractability can be encountered [Rajamoney and DeJong 87]:

- Large Search Space Problem - This occurs when the number of rules in the domain theory that *appear* to be relevant in deriving a proof is large. Finding the appropriate subset of rules that actually link the input example to the concept might involve exploring paths that appear promising but that do not lead to the creation of a proof. If explanation generation is viewed as search, then in the worst case the size of the space explored is B^D, where D is the maximum length of an explanation and B is the maximum number of rules applicable at any point. The large search space problem occurs when B^D is large, and is characterized by large values for D and/or B.

- Small Links Problem - This occurs when the number of rules required to form a single proof is too many for the system to construct the complete proof within the allotted resources. This problem is partially independent of the large search space problem and may occur even when no search is involved. Viewing explanation generation as search, it is characterized by large values for D.

The large search space problem is addressed by [Lebowitz 86a; Lebowitz 90] as well as by Ellman [Ellman 87; Ellman 88a; Ellman 88b; Ellman 89b]. Lebowitz uses SBL generalizations to focus on features that would appear specifically as causes or effects in an explanation. This constrains the rules that must be considered. Ellman introduces the notion of approximating the domain theory to create simpler explanations. Ellman additionally considers the problem of small links. [Utgoff and Saxena 88] focus on a somewhat different problem, the efficient use of definitions learned by EBL.

2.5.1.3. The Problem of Representational Inadequacy for Similarity-Based Learning

The entities that can be described by the input and output for a learning system are determined by their knowledge representation. Thus the choice of a particular representation language is said to bias SBL in that it constrains the concepts output from — and therefore learned by — the method.

Preferably the creator of a learning system should not be required to anticipate all input that the system will encounter in a given domain. If that were necessary, learning systems would be limited to small, non-complex domains. Furthermore, it is desirable that a given learning system be applicable across a wide range of domains. But often a representation suited to one domain does not adequately capture the objects in another. If the burden of anticipating all future processing of a learning system is to be removed from the initial design, a learning system must be able to modify, or at least add to, its body of representable objects.

[Flann and Dietterich 86], as well as [Stepp and Michalski 86] and [Watanabe and Elio 87], address the issue of modifying a representation in order to use the best one available for a particular context. Essentially, they use a domain theory to translate one representation into another. [Utgoff 86] addresses the problem of learning new vocabularies.

2.5.2. Related Work in the Integration of Deduction and Induction

Other work in machine learning might be viewed as integrating inductive, or similarity-based, and deductive, or explanation-based, mechanisms. A number of researchers have investigated the use of **analogy** as a learning mechanism. Analogy, the inference that if two or more things agree in some respects they will probably agree in others, involves the matching of one entity with another. Thus the matching techniques of SBL are often applied here. Furthermore, since similarities might be found at a different level from the purely superficial one of entities' observable characteristics, the entities are often analyzed using deductive, or explanation-based, mechanisms as well. Examples of work in analogy include [Burstein 86; Carbonell 83a; Carbonell 83b; Kedar-Cabelli 85; Russell 86].

Exemplar-based learning is another mechanism that combines inductive and deductive techniques. In exemplar-based learning a concept is represented by a stereotypical example. Deduction, or explanation derivation, is used to determine whether an input might be an instance of the concept even if it does not exactly match the definition of the exemplar for the class. An example of work in this area is [Bareiss and Porter 87]. More generally, work has been done in **Case-Based Reasoning (CBR)**. The philosophy of CBR is that solutions to problems need not always be derived from scratch. A new problem solving situation might be similar to one encountered earlier. It might be useful to modify the solution to the old problem. In CBR cases of previously seen problems and their solutions are stored in memory. When a new problem arises an old case is retrieved. Mechanisms similar to those described above for analogy may be used to retrieve appropriate cases for modification. Work in CBR includes [Hammond 88; Kolodner 87].

2.6. Summary

This chapter has surveyed work in machine learning related to our research. There are two major goals of our research: (1) to address the problem of incompleteness of the domain theories used by explanation-based learning and (2) to address the problem of providing focus for an inductive similarity-based learning system. As EBL and SBL are complementary in nature, our solution involves an integration of the two.

Previous solutions to the problem of domain theory incompleteness rely either on an auxiliary theory or on an outside expert to find missing knowledge. If an auxiliary theory is expected to be perfect, it suffers precisely from the problem addressed. Our solution, implemented in Gemini, makes use of an additional theory of heuristics, but the theory acts only as a guide and is therefore not required to be perfect. It makes use of general knowledge inherent in the structures of explanations, rather than referring to any particular domain semantics that might limit its applicability. Gemini does not require interaction with a user and can therefore function autonomously.

Previous solutions to providing focus for SBL suffer from problems analogous to those for incompleteness. Some, such as Mitchell, Salzberg, and Swaminathan, make use of a domain theory, but the theory is treated as if it were perfect. On the other hand, Van Lehn relies on an outside source to provide the focus. Gemini makes use of two knowledge sources, both internal to the system. One is a partial domain theory, and the other is a theory of heuristics that indicate a variety of ways in which domain theory knowledge can be used.

Chapter 3

Gemini: An Integrated Learning System

3.1. Introduction

Explanation-based and similarity-based learning techniques have been applied with success in a number of domains. In addition, since the two methods are complementary in nature, researchers have found it natural to use one as a solution to any problems with the other. In the previous chapter we described a number of machine learning systems integrating EBL and SBL in this way. In this chapter we describe Gemini, an integrated learning system that combines EBL and SBL in a feedback loop, allowing intercommunication for the mutual benefit of the two methods.

Two major criticisms were raised in Chapter 2 with respect to previous solutions to the problem of theory incompleteness in EBL: (1) the requirement that auxiliary theories be complete (e.g., [Rajamoney 89; Pazzani 88; Wilkins 88]); (2) the need for active presence of an expert (e.g., [Hall 86; Kodratoff 87; Wilkins 88]). Similarity-based learning plays a role, but this role is limited to verification or generalization of knowledge proposed by another source. Our solution to the problem of incompleteness places greater emphasis on SBL as the primary mechanism for acquiring missing knowledge. In Gemini, SBL makes use of a knowledge base of heuristics that guide it in: (1) selecting examples over which to perform induction, and (2) focusing on particular features of the examples considered. Although the knowledge base of heuristics is, in a sense, an auxiliary theory, it is a general theory based upon the tree-like structures of explanations, or proofs, rather than on any particular type of domain. As the heuristics are treated only as a guide, completeness is not required. The origin and use of heuristics are discussed in Chapter 4.

Discussion in Chapter 2 on providing focus for similarity-based learning concentrated on two basic approaches: (1) felicity conditions between a teacher and the learning system [Van Lehn 87] and (2) the role of explanations in determining relevance of features [Salzberg 85; Swaminathan 88]. Since a goal of our work is to minimize interaction with an expert or teacher, we concentrate on the latter approach. In addition to focusing on specific features, however, our heuristics make use of explanation information to guide SBL's selection of examples for induction.

Examples used to illustrate the points in this chapter are taken primarily from the network fault diagnosis domain introduced in Chapter 1. Other examples are taken from the domain of terrorist event news stories, described in more detail in the following section. Section 3.3 introduces the learning model behind Gemini, while Section 3.4 presents the architecture implemented. EBL and SBL are implemented in Gemini as separate phases. Section 3.5 describes EBL as implemented in Gemini. SBL is used in two distinct ways: as conceptual clustering in order to construct a memory of related inputs (see Section 3.6), and for learning concepts from examples in order to augment a potentially incomplete rule base used by EBL (Section 3.7).

3.2. Terrorist Event News Stories: A Second Domain

The analysis of terrorist event news stories is among the domains that have been implemented in Gemini. This domain is modeled loosely around the following performance task: Given information from a news story describing a terrorist event, suggest an action that a law enforcement agency might take in response, given a database of earlier events. One might conjecture that in order to devise a proper response, a planner would require the most pertinent information about a specific case, as well as information about related occurrences. EBL provides focus on the most relevant aspects of an input, while the conceptual clustering style of SBL organizes input cases together with others considered to be most similar.

As an example, consider the following excerpts from New York Times news stories describing acts of terrorism in 1986:

1. PARIS, Feb. 3 - An explosion, apparently caused by a bomb, ripped through a crowded shopping arcade on the Champs-Elysees tonight. Eight people were wounded, three of them seriously. The blast ... sent shards of glass flying through the Hotel Claridge, filling it with smoke. Witnesses

> said that damage was extensive... The blast, which took place at about 9:30 P.M., occurred in one of Paris's most popular nighttime entertainment areas...

2. PARIS, Feb. 4 - A bomb ripped through a crowded bookstore in the Latin Quarter tonight, wounding four people, the police said. ... The bomb that exploded tonight was planted in the basement record section of the Gibert Jeune bookstore on the Place St. Michel. The blast occurred at 7:40 P.M. with scores of customers in the store.

Gemini is given as input a conceptual description of these events (though it would be reasonable to parse the natural language input [Lebowitz 80]). The conceptual description is a hierarchical frame, as will be discussed in Section 3.5.2. Using a built-in domain theory about terrorist events, EBL derives explanations that indicate that relevant information for each event included the fact that it was a bombing and that it took place in a crowded location. The specific time of occurrence, while related to the fact that the target was crowded, would not be considered to be important. SBL would organize these events closely together in memory as their descriptions are similar. The use of explanations for similarity-based learning in this domain is described in [Danyluk 87].

3.3. The Learning Model

Gemini integrates EBL and SBL as follows: EBL analyzes an input example by explaining it and by providing SBL with additional information about the features describing the input. For example, only a subset of the features of an input instance would appear in an explanation. SBL could assume that these were more important than those that did not. In general, an explanation, or even a partial explanation, sets a context in which the features describing an input instance may be viewed. SBL can be used in augmenting a potentially imperfect domain theory used by EBL. In Gemini, SBL is used to learn information missing from a domain theory. The integrated learning model is illustrated in Figure 17.

Figure 17: Learning Model Integrating EBL and SBL

3.4. Gemini Architecture

Gemini is an integrated learning system composed of three major subsystems, one performing explanation-based learning (G-EBL) and two carrying out different types of similarity-based learning. One form of SBL carried out by Gemini is a type of conceptual clustering that creates a hierarchically organized memory of inputs seen by the system (G-CC, or Gemini Conceptual Clustering). The second type of SBL carried out by Gemini is learning from examples in order to fill gaps in partial explanations produced by EBL (G-RI, or Gemini Rule Induction).

Gemini's architecture is shown in Figure 18. Input to G-EBL is the name of a goal concept and an example of that concept. G-EBL attempts to construct an explanation, or proof, describing why the example is an instance of the goal concept. Built into the system is a domain theory in the form of a rule base that is used by G-EBL in constructing explanations. Gemini does not assume that the domain theory is complete. It is possible that for any given input, Gemini will be able to construct only a partial explanation (or none at all).

G-RI constructs new rules when a gap is detected in the domain theory used by

Figure 18: Gemini Integrated Learning Architecture

G-EBL. During explanation construction, a gap is detected when it is known that a subgoal must be inferred, but it is not possible given the rules in the domain theory (i.e., the domain knowledge is overly specific). G-RI treats the uninferred subgoal as the name of a concept to be learned. The input example is assumed to be a positive example of that concept. In order to create a general concept definition, additional examples, both positive and negative, are retrieved from Gemini's processing history, along with their derived explanation information. The examples are found in a memory (the GBM, or generalization-based memory) that is constructed by G-CC. The learned rule is generalized by invoking heuristics that indicate ways to remove unnecessary information based upon contextual information extracted from the (perhaps partial) explanations constructed by G-EBL.

All three major components of Gemini share access to a static semantic memory that defines hierarchical relationships among entities in a given domain. More specifically, the static semantic memory is a directed acyclic graph representing relationships among feature values of input instances. Details about the static semantic memory are given later with respect to the manner in which it is used by Gemini's procedural components. G-EBL is discussed in detail in Section 3.5. G-CC and G-RI are described in Sections 3.6 and 3.7, respectively.

3.5. Explanation-Based Learning in Gemini

3.5.1. Overview

This section discusses Gemini's explanation-based learning component. We begin by describing the inputs and outputs of G-EBL in Sections 3.5.2 and 3.5.3. Additional knowledge required by, and built into, the system is given in Section 3.5.4. We follow this with a description of the EBL algorithm implemented in Gemini in 3.5.5. Section 3.5.6 gives a summary of the method and a brief comparison to other EBL algorithms.

3.5.2. Input

Input to Gemini's EBL component is a pair consisting of a goal concept and an example which is an instance of that concept. The concept instances are represented as hierarchical frame structures. These are frames in which slot fillers describing features may be values, pointers to subframes that describe features in more detail, or nil, which indicates that a value is not known.

A typical goal concept in the domain of network fault diagnosis is a problem diagnosis, such as *poss-fiber-optic-cable-prob* (a possible fiber optic cable problem). A corresponding input frame (i.e., the instance) represents the state of the network at the time of diagnosis. The sample input frame in Figure 19, for example, represents the state of a network with a possible fiber optic cable problem. Although Gemini's EBL component can receive as input a hierarchically organized frame, the input for this domain is a frame at a single level. Some slots give the status of particular parts of the network. For example, the slot *cond-tgt-node-responds-to-ping?* indicates whether the target node of an initiated communication responds (i.e., whether it appears to be ‘‘alive’’). Other slots give descriptive information. For instance, the slot *td-the-user-is-on-an-Ethernet?* indicates whether the user initiating a failed communication is on an Ethernet. All slot values in this domain are either true (T) or false (F). Nil would indicate that a slot value was unknown. The input frame in Figure 19 is shown with the actual slot names and values used by Gemini. Slots for this domain were extracted from a prototype expert system for network fault diagnosis developed at CMU [Eshelman 88]. They correspond to questions asked of the user directly, rather than inferences made by the system.

Figure 19: A Sample Input Frame for Network Fault Diagnosis

A second example is taken from the domain of terrorist event news stories. Here, actual news stories are transcribed into hierarchical frames. A part of the frame representation for the second news story described earlier in Section 3.2 is shown in Figure 20. In this example, slots provide information extracted from story #2. For instance, information is given about the type of event and its cause. In this domain, slot values are more descriptive than the binary values used in the network fault diagnosis domain. A number of slot values are pointers to more in-depth information about a particular feature. For example, time information includes a full date, a time of occurrence based upon a 24-hour clock, and information about the difference in time from Eastern Standard Time.

Figure 20: A Sample Input Frame for Terrorist Event News Stories

Every input frame is given to Gemini as an instance of a specific goal concept. In the terrorist event domain, the goal concept is always taken to be the constant *terrorist_event*. EBL is used to explain why the action is a terrorist event, as opposed to some other type of news event.

3.5.3. Output

The output of Gemini's explanation-based learning component is an explanation (perhaps partial) describing why the input example is an instance of the goal concept. The explanation generated by Gemini linking the network fault given in Figure 19 to the diagnosis of a possible fiber optic cable problem is given in Figure 21. In the network fault domain, *user* refers to a user of a network who has attempted to communicate with another node in the network, but has failed. The explanation tree in Figure 21 may be read as follows:

> If the target of the failed communication responds in general, but the user who initiated the failed communication is not able to reach any node in the network, then the problem appears to lie in the source of the failed connection; if this is the case, and the user is on a token ring network, then the problem appears to be in the connection of the user to the token ring network; if this is true, and...

The explanation generated by Gemini linking the terrorist event input example of Figure 20 to the goal concept of a terrorist event is shown in Figure 22. The explanation tree may be read as follows: (Numbers in the text refer to the corresponding assertions in the figure.)

> (1) If a public place is open with no warning of an attack, then the place will be crowded. (2) If a public place is crowded and massive force is used against that place and no specific target is named, then the general public will be alarmed. (3) If an attack is made to frighten the general public, then the attack constitutes a terrorist event.

Figure 21: A Sample Explanation for Network Fault Diagnosis

3.5.4. System Knowledge

In order to construct Gemini's explanations, G-EBL requires a domain theory, which in Gemini is a knowledge base of *if-then* rules. The conditions of a Gemini rule refer to specific input example features and/or (sub)goals that must be proven. Conditions referring to actual input features contain variables to enable the specification of relationships between the input example entities; all variables are implicitly universally quantified. A (sub)goal is a predicate to be proven true of the input example as a whole. The conclusion of a Gemini rule consists of a single predicate. In the current implementation, no variables appear in the conclusion of a rule, as it is defined to be implicitly true of the input example as a whole.

Figure 22: A Sample Explanation
for Terrorist Event News Stories

As an example, consider a rule from the network fault diagnosis domain that states ‘‘If a user is complaining that a connection cannot be established with a particular destination, but the same user is able to reach other nodes on the network, then the problem appears to lie with the destination of the unestablished connection.’’ Recall from Chapter 1 that the knowledge for this domain is encoded on a level that enables isolation of the fault to a particular segment of the network, but does not allow deep analysis of the problem. The conditions of this rule refer to individual feature values of the fault state description, specifically the values of the features *complaint-user-cant-connect?* and *td-can-user-reach-another-net?* shown in Figure 19. The conclusion refers to the faulty network state, the input example as a whole.

G-EBL shares access with SBL to a static semantic memory that defines hierarchical relationships among entities in a given domain. More specifically, the static semantic memory is a directed acyclic graph representing relationships among feature values of input instances. It is essentially a semantic net in which edges are restricted to the ISA relationship. As feature values for the network fault diagnosis domain are restricted to true and false, no static semantic memory is needed to define relationships among those values. A static semantic memory is useful, however, for the terrorist event domain. A fragment of that memory is shown in Figure 23.

Figure 23: Fragment of Static Semantic Memory for Terrorist Event News Stories

The static semantic memory allows Gemini's rules to concisely refer to classes of values. Consider the following rule from Gemini's domain theory for terrorist event stories:[9]

```
IF
    ISA (?X, PLACE) ∧
    GEN (?X, TYPE, PUBLIC_PLACE) ∧
    HAS (?X, AVAILABILITY, OPEN) ∧
    ASSERT (NOT_SIGNIFICANT_WARNING)

THEN
    ASSERT (CROWDED_TARGET)
```

This rule states ‘‘If the place of occurrence is public and is open when the event occurs, and Gemini can assert that there was no significant warning, then infer that the target is crowded.’’ Here, a condition of the rule refers to any public place. Whether the place specified in the input frame is a specialization of this concept can be inferred from the static semantic memory. This would otherwise have to be done by creating a separate rule for each possible specialization of public place.

3.5.5. Gemini’s Explanation-Based Learning Algorithm

Gemini’s explanations are generated by backward-chaining from the input goal concept. Given a goal concept, Gemini finds all rules in its domain theory from which the goal could be inferred in one step, i.e., having the goal concept in the conclusion. Gemini then evaluates the premise of each of these to see if the conditions have been met. This may require checking the input frame for specific feature values, or it may require further backward chaining from a subgoal. If all conditions are met, an explanation has been found. If conditions referring to specific feature values are not met, a reasoning path fails, and another one is attempted, if one exists. Gemini’s explanation generation procedure is essentially depth-first search.

As Gemini’s domain theory may be incomplete, Gemini will sometimes fail to generate a full explanation. (Partial explanations will be shown in Section 3.7 on the induction of missing rules.) This may occur for one of two reasons, corresponding to the two manifestations of incompleteness described in Chapter 1. A rule base is incomplete if:

---

[9]Gemini’s rule syntax has been modified slightly here for illustrative purposes. See Appendix B for detailed domain specifications.

1. complete rules are missing, or
2. conditions on a rule are too specific and will therefore not allow the rule to be applied to all relevant cases.

As disjunctive conditions are represented as separate rules in Gemini, the first of the above two incompleteness types may result in either of the following:

- There are *some* missing disjuncts — i.e., the domain theory has rules for making the necessary conclusion, but the set of rules is too specific, or
- There is *no way* to infer a particular subgoal given the domain theory — i.e., all disjuncts are missing. The subgoal to be proven cannot be found in the conclusion of any domain theory rule.

The second of these is easy to detect, and the detection is implemented in Gemini. All other types of incompleteness require consideration of the rule - or rules - that were applied, but failed. Any of these is potentially responsible for the failure of the explanation. One promising approach to the problem of abduction - or selection of a ‘‘best’’ (partial) explanation - is described by [Fawcett 89] and [Ng and Mooney 90], among others. It involves ranking all possible (partial) explanations according to heuristic criteria such as:

- the conciseness of the explanation (‘‘shorter is better’’);
- the number of features referenced in the explanation (‘‘greater coverage is better’’).

Gemini has largely been developed as a testbed for evaluating rule induction heuristics as described in Chapters 4 and 5. We provide a control on our experiments in the following manner: A full explanation is generated, and then the rule to be learned is removed from the explanation *by hand*, making it incomplete. We then test the system to see what contextual heuristics are useful in learning the missing rule. Thus, at this point, Gemini does not generate partial explanations on its own.

3.5.6. Summary

This section described Gemini's explanation-based learning component, specifically discussing its inputs, outputs, and built-in knowledge. Input to G-EBL consists of a predicate describing a goal concept and an example of that concept. Input examples are represented as hierarchical frames. G-EBL learns by constructing an explanation, or proof, describing why the input example is an instance of the goal concept. In order to construct explanations, G-EBL requires a domain theory, which in Gemini is a knowledge base of rules. G-EBL also shares access with the SBL components to a static semantic memory that defines relationships among feature values of input instances.

It should be noted that Gemini does not perform the generalization step of many other EBL systems. That is, it does not create a generalization of the input that is consistent with the explanation structure derived. This is not due to any deficiency in Gemini. As it is of secondary importance to the primary goal of extracting contextual information from explanations in order to complete those that are partial, it simply has not been implemented at this time. The benefit of not performing generalization is a saving in the running time of the algorithm.

3.6. Building a Memory of Input Examples in Gemini

3.6.1. Overview

This section discusses one of the two distinct types of similarity-based learning performed by Gemini: the conceptual clustering style of learning that has as a goal the organization of system inputs according to their similarities. What is being learned here are various relationships among the input entities. This is of use to the second type of SBL performed by Gemini, which applies a variation of learning from examples in order to formulate rules missing from the domain theory used by EBL.

We begin with descriptions of the inputs and outputs of Gemini's memory organizing component in Sections 3.6.2 and 3.6.3. Additional knowledge required by, and built into, G-CC is given in Section 3.6.4. We follow this with a description of the algorithm used to discover similarities among input examples so that they may be organized into a memory of the cases seen. We conclude the section with a summary of the method and a comparison to related work.

3.6.2. Input

G-CC receives Gemini's input frames as its primary input, and, in addition, is given their corresponding explanations. Recall that before entering the clustering component of the system, an explanation, either partial or complete, has been constructed for an input example by G-EBL. This provides additional information about the features' relative importance.

Similarities among the features of the input frames play a large role in determining their relationships. However, if they were used as the sole factor, then Gemini would have no means by which to distinguish between similarities that were truly significant and those that were only coincidental. As an example, consider three possible scenarios from the terrorist event news story domain. Say that one story reported an assassination attempt on the leader of country X. A second told of a bombing in the home of that same leader. A third story reported of a bombing in the home of a movie star. Given feature similarity as the primary measure of relatedness, a system might determine that there was a relationship between the first two events, since the targets in the respective events were the same. However, it might also believe that the second two were related, as both were bombings in the homes of famous people. Clearly, people find a qualitative difference between the levels of similarity. Explanations generated by G-EBL can help focus on specific sets of relevant features.

3.6.3. Output

Relationships among input examples are recorded in a **Generalization-Based Memory** (GBM) [Lebowitz 80; Lebowitz 86b] that is the output of G-CC. In general, a generalization-based memory is a directed acyclic graph in which terminal nodes are specific instances; internal nodes represent generalizations that classify the input instances that are their children. As an example, consider the small fragment of a GBM shown in Figure 24. This example is taken from the domain of network fault diagnosis.

For illustration purposes the input frames shown here have been abridged from the actual representation. We additionally include in Gemini's GBM explanatory information derived for the input frames. That is excluded here for notational convenience. Leaves of the structure shown in Figure 24 describe input examples to Gemini. One of them, as indicated, describes a faulty source node, where that node is on an Ethernet. (It describes a faulty source, as the destination of the unestablished connection appears to be responding properly, while the source is not.) It is classified together with one other example. Their parent node, labeled as a representation for a source problem, describes the features common to its children.

Figure 24: Fragment of a Memory of Network Faults

The fragment of memory shown in Figure 24 is a binary tree. However, this is not always the case. Most generally, as indicated above, the GBM constructed by G-CC may be any directed acyclic graph. Consider the memory fragment shown in Figure 25, taken from the domain of terrorist event news stories. Again, frame representations have been abridged here for purposes of illustration. Figure 25 illustrates a scenario similar to that described at the beginning of this section. Here two stories are alike in that they have similar target locations. One of them, however, is also similar to another in that these two share the terrorist method. Note also that the first two stories are found to be similar in spite of the fact that their locations are not identical. Various degrees of similarity may be specified as acceptable within Gemini.

Figure 25: Fragment of a Memory of Terrorist Event News Stories

As mentioned briefly in Section 3.6.2, a structure such at that given in Figure 25 might not adequately distinguish between significant similarities and spurious correlations. Explanations (complete or partial) provide additional information about the relative importance of various features. Those actually playing a role in an explanation might be considered more relevant, and thus more significant, than those that do not. For example, if a more specific target - say, a person - had been indicated in the first two stories of Figure 25, then features about the person would presumably play a role in the respective explanations. The actual method of attack might not appear as a significant feature in these cases. Now say that in the third event, there was no specific target known. Here the method of attack might appear as a feature in an explanation. The difference in emphasis on various features might result in a memory such as that shown in Figure 26, where the third event is not related to the others at all.

Figure 26: Memory Amended after Additional Explanatory Information

3.6.4. System Knowledge

As stated in Section 3.5.4, Gemini's SBL components share access with G-EBL to a static semantic memory that defines hierarchical relationships among entities in a given domain. It is this knowledge structure that enables Gemini to find similarities such as the one pictured in Figure 25. In that case, two event locations were considered to be similar as they were both in the state of New York, even though they were actually in different cities. Given two feature values, Gemini is able to determine whether they are related by searching the static semantic memory for a common generalization of the two. If there is none, then they are not considered to be similar.

If there is a common generalization, the closeness of the similarity must be determined. In Gemini this is a function of the distance from the feature values to their generalization in the static semantic memory. Specifically, it is the sum of the lengths of the shortest paths from each value to the generalization. For example, in Figure 23, the most specific generalization of *movie_theater* and *flea_market* is *entertainment_place*. Their distance measure is 3, the total number of edges traversed to reach the generalization. Note that different segments of the static semantic memory might have been defined at different levels of granularity. The number of edges between concepts does not necessarily describe conceptual similarity consistently. To overcome this, we have implemented a facility that allows for more details to be given about each concept in the hierarchy. The similarity of two concepts is measured by both the distance to a common generalization and the similarity of their more detailed descriptions.

Now consider the fragment of a static semantic memory shown in Figure 27. As it is a graph, rather than a tree, it is possible for two values to share multiple common generalizations. In such cases additional contextual information is useful in selecting the generalization that is most appropriate. Say, for example, that two events had taken place at the hours of 6:00 PM and 8:00 PM, respectively. These times of occurrence are similar in that they are both *late_evening* hours. However, they are also *business_hours* for places of entertainment such as movie theaters and restaurants. Now say the two events had taken place at movie theaters, presumably crowded at the time of the attack. As the events took place in a business context, the most informative generalization for the two time values would probably be *business_hours*, rather than *late_evening*, another valid generalization. In general, the presence of certain features, either alone or in combination, indicates a context in which to view other features. In Gemini, this type of context information is explicitly given in the rule base for a domain. Consider the following variation of the rule given in Section 3.5.4:[10]

```
''If an event occurs at a place of business and
     the place is open at the time of the attack,
  Then the place of occurrence is crowded.
  {Mark time with *business* context}''
```

If this rule applies to the input, it explicitly causes all time-related input features to be generalized within the business context. Thus, in addition to the static semantic memory, the memory building component of Gemini requires that context information be indicated in the rule base used by G-EBL. This idea of using rules to describe feature context is unique to Gemini.

[10]This rule has been rewritten in English for clarity.

Figure 27: Fragment of Static Semantic Memory with Context Markers

3.6.5. Learning Algorithm

Gemini builds its memory of input examples ‘‘from the bottom up’’, considering the relationships among all input examples. The algorithm is incremental, comparing an input frame against each example already in the memory. This pairwise matching results in many possible generalizations, from which a ‘‘best’’ generalization is selected. Since the pairwise matching of input frames is the basis of the clustering algorithm, Section 3.6.5.1 discusses this in detail. The clustering algorithm itself, which makes decisions about the generalizations to retain, is presented in Section 3.6.5.2.

3.6.5.1. Pairwise Matching of Gemini Input Frames

The two preceding sections describing the inputs and outputs of G-CC introduced the notion that two input frames may be similar in a number of ways. Among these are the following:

- They may share a large subset of features.
- They may share features that, while not identical, can be described by more general values.
- They may share features that are similar when viewed in the same context.
- They may have identical - or nearly identical - corresponding explanations.
- They may share a large subset of relevant features, where relevance is determined by the explanations.

In implementing G-CC, we have taken a number of these into consideration. The algorithm for pairwise comparison of input frames is given in Figure 28. Input to the algorithm are two frames (Frame1 and Frame2) with their corresponding explanations (Explanation1 and Explanation2). The algorithm builds a third frame that is a generalization of the two inputs. When an input frame enters the system, it is matched against each input already seen, using the pairwise matching algorithm. Later, a single best match is selected based upon scores given to the generalizations.

The first phase of the algorithm compares the frames for similarity of their feature values. For each slot, the values in Frame1 and Frame2 are compared. Exact matching is attempted before any generalization is performed. If two slot values match exactly, the corresponding value in Frame3 is the same. If not, generalization via the static semantic memory is attempted. Edges from the slot values to a generalization are counted. Smaller distances yield better generalizations. A single best generalization is selected as the value for Frame3. For example, if two slots had the values *store* and *flea_market*, the generalized value for Frame3 would be *shopping_place*, rather than a value higher in the hierarchy of Figure 23. It is possible that Gemini will be unable to find any relationship between certain pairs of slot values. If so, these are indicated in Frame3 as unknowns.

```
INPUT:  Frame1, Explanation1, Frame2, Explanation2.
OUTPUT: Frame3 - with annotations for explanation information,
        match statistics.

PHASE I - Determine feature set similarity.

For each slot in Frame1 or Frame2
   * Create a slot of that name in Frame3;
   * IF the value of that slot name is unknown in either
     Frame1 or Frame2
        THEN    /* no slot match can be established */
            the value in Frame3 is unknown;
            the match score for the slot is unknown
   * IF the values in Frame1 and Frame2 are identical
        THEN    /* exact match */
            the value in Frame3 is the matched value;
            the match score for the slot is 0
   * IF the values in Frame1 and Frame2 have common generalizations
        THEN
            select at most one generalization - context markers
            indicate highest priority;
            the value in Frame3 is the generalized value;
            the match score for the slot is determined by distance
            (number of edges) between the two initial values
   * IF no match
        THEN    /* treat the same as an unknown */
            the slot value in Frame3 is unknown;
            the match score for the slot is unknown
Generate final count of unknowns in Frame3;
Generate final total of slot match scores in Frame3.

PHASE II - Determine explanation similarity.

Find the intersection of Frame1 explanation features and Frame2
   explanation features.
```

Figure 28: Pairwise Input Frame Comparison

The first phase of the algorithm results in a new frame that is a least generalization of the two inputs. In addition it produces a measure of the quality of the match. Result frames with lower total match scores are more similar, and thus better. Result frames with smaller numbers of unknowns are more specified and are thus better as well. Often there is a tradeoff between the match score and the number of unknowns. In Gemini the match score takes higher priority with the stipulation that the number of unknowns be below a reasonable threshold.

The next phase of the pairwise matching algorithm considers similarity of explanations. Although the explanations could be compared in a number of ways, we have found that it is effective to simply consider the sets of input features seen in the explanations. We refer to these as explanation features. Given two sets of explanation features, this step of the algorithm takes their intersection and stores that with the new generalized frame.

Once two frames are compared and their generalization is created, G-CC must determine whether the match is sufficiently good to be noted in the memory. This is accomplished by considering three factors:

1. the total match score
2. the total number of unknowns / complete mismatches
3. the percentage of explanation features matched.

All three of these must be below pre-specified thresholds.

3.6.5.2. Gemini's Clustering Algorithm

The complete algorithm for building a memory of input examples is given in Figure 29. When a new input frame is to be incorporated into the memory, it is compared pairwise against each previously seen example as discussed above. A single best match is then selected, where best is determined by selecting the generalization with the least total distance computed during slot matching.[11] If the best match is an acceptable one, the generalized frame is stored in the memory as a parent of the two frames that generated it. Generalizations are compared similarly to create deeper hierarchies.

As even the best match found for an input frame might be unacceptable, parts of the memory might become disjoint. That is, the memory might be a set of graphs rather than a single connected structure. Conceptually, one might view the graphs as entities unrelated except for the fact that they are taken from a single domain.

---

[11]Earlier implementations of Gemini did not select a single best generalized frame, but rather saved all acceptable generalizations. This resulted in an exponential time complexity. We have found that, in spite of its apparent restrictive nature, the updated algorithm creates memories that seem no less useful for later rule induction.

```
INPUT:  example i, with corresponding explanation.
OUTPUT: Generalization-Based Memory.

ALGORITHM:

1.  For every terminal node n in the existing memory
       * Compare the input example i against n;
       * Find a measure of their similarity, considering both degree
         of match between their feature sets and degree of match
         between their corresponding explanations;
       /* i.e., perform the procedure given in the previous figure */

2.  Select the best match best (Better distance measures have
    highest priority);

3.  IF best falls within acceptable bounds for retaining the match
       THEN
           make i a terminal node in the GBM;
           create a new internal node g that is a generalization
              of i and the best matching n;
           create arcs from g to its specializations i and n
       ELSE
           make i a terminal node in the GBM, but do not attach it
              to any existing part of the GBM.

4.  Recursively invoke the algorithm: Treat generalization g as input;
    treat generalizations one level up from leaves as leaves.
```

Figure 29: Gemini's Clustering Algorithm

3.6.6. Summary

This section described one type of similarity-based learning performed by Gemini, in which the system's goal is the generation of a memory that indicates relationships among Gemini's input examples. That information can later be used in selecting examples from which to perform induction as described in Section 3.7.

The concept of building a memory of input examples for later use is not new. Memories of this type were constructed previously in systems such as UNIMEM [Lebowitz 87], IPP [Lebowitz 80], and CYRUS [Kolodner 83], among others. There are a number of differences between our implementation and the others. The major differences result from the fact that new input examples enter our memory from the bottom (i.e., with a comparison against *all* other input examples), while, in other systems, examples are placed in memory following a path from the most general concept down to the leaves. Thus in Gemini the placement of a new example is less dependent upon the order of previous inputs seen. A second distinction between our method and others is our use of information extracted from explanations. G-CC does, however, also have a greater time complexity, requiring $O(n^2)$, where n is the number of input frames, in the worst case to incorporate an instance, while UNIMEM, for example, requires only $O(\log n)$.

Gemini's memory building is also similar in spirit to work in the area of **Case-Based Reasoning** (CBR), most notably Kolodner's [Kolodner 87]. The underlying principle of case-based reasoning is the idea that in problem solving it is often most effective to recall a solution to an earlier problem and to modify it, rather than to derive a new solution entirely. Building a memory provides a mechanism for finding and maintaining similarities.

3.7. Rule Learning in Gemini

3.7.1. Overview

During the course of deriving an explanation, or proof, that an input example is an instance of a particular concept, the EBL component of Gemini may find that it is unable to complete the proof. Assuming that there is no noise in the input and that the domain theory is not incorrect, this indicates that the domain theory is incomplete. Recall that, although we assume correctness and tractability of the domain theory used in G-EBL, we do not assume that it is complete. A gap in Gemini's rule base is detected by EBL when it is unable to deduce a subgoal that would have to be true for an explanation to be complete. As an example, consider the partial explanation shown in Figure 30. This is almost identical to the explanation shown earlier in Figure 21, except that one of its subgoals, *looks-like-source-prob*, has not been inferred. This might occur in one of two ways. First, it would occur if the domain theory contained no rules with *looks-like-source-prob* in their conclusions. It might also occur if there were disjunctive ways, but no disjunct was provable. The task of the SBL component discussed in this section is to determine the relationship between the uninferred subgoal and the input features (and/or inferred subgoals) that would imply it. The task is to learn a rule which, when instantiated, would not only complete the current explanation, but would be correctly applicable to future cases as well.

Figure 30: A Partial Explanation
Derived by Gemini

The rule induction component, G-RI, uses a variation on methods of learning from examples in order to construct rules missing from the rule base. We begin with a description of the inputs and outputs of G-RI in Sections 3.7.2 and 3.7.3. Additional knowledge required by, and built into, G-RI is given in Section 3.7.4. We follow this with a description of the algorithm used to learn missing rules.

3.7.2. Input

Gemini's rule induction algorithm receives two principal inputs:

1. the input frame, or example, for which a complete explanation could not be constructed;
2. the best partial explanation derived for the example.

As discussed in Section 3.5.5, the problem of selecting a best partial explanation is a difficult one, and is not addressed in this research. If, during generation of an explanation, Gemini encounters a subgoal for which no rules exist in the domain theory, that partial explanation is selected as the one to be completed. Otherwise, selection of a best explanation is currently performed by hand.

3.7.3. Output

G-RI creates a new rule for G-EBL's domain theory. The generation of a missing rule involves the determination of both its premise and conclusion. When G-EBL is unable to complete a proof, it is because at least one necessary subgoal is not provable using the partial theory, as illustrated in Figure 30. In that example, to make the inference *looks-like-source-prob*, G-RI must find the set of input features that correctly imply it. The conclusion of the missing rule is the unproven predicate. The conditions must be extracted from the input example. This is illustrated in Figure 31.

Figure 31: Syntax of Missing Rules

A missing rule will not always conveniently fall at the leaves of an explanation tree. Gemini may fail to prove a subgoal higher in the tree, as in Figure 32 a. In this case, the conditions of the rule may refer to additional subgoals. Since Gemini generates explanations by chaining backward from the goal concept, it stops as soon as a subgoal cannot be proven. That is, it does not continue generating the ‘‘lower’’ parts of the explanation. In Figure 32 a we include the lower part of the partial explanation simply to illustrate the missing rule’s context. Figure 32 b shows Gemini’s view of the partial explanation.

The G-RI algorithm learns a single rule at a time. This does not, however, restrict learning to partial explanations with only one gap. If an input example’s explanation is missing multiple rules, each rule is learned individually as follows: For *each* unproven subgoal:

1. Take the input example to be a positive instance of that subgoal;
2. Given the example and its partial explanation (however shallow it may be), use contextual heuristics as usual to retrieve other positive and negative examples from the GBM for learning.

Since the partial explanation in this situation is missing multiple inferences, it will presumably contain less contextual information than a more complete explanation. In general, more examples will be required to fully generalize the learned rule. A future enhancement to Gemini might consider knowledge about multiple missing rules as additional contextual information in and of itself.

3.7.4. System Knowledge

The problem of learning a rule to fill a gap in an explanation is treated as an instance of concept learning from examples. Specifically, Gemini treats an uninferred subgoal as a concept to be learned. The input example for which the partial explanation was derived is assumed to be a positive instance not only of the final goal concept, but of the subgoal as well. Gemini’s task is to select those input features (or other subgoals inferred from them) that best describe the uninferred subgoal. If there are disjunctive ways to do so, only the disjunct covering the given example is learned.

In order to find the missing premise for a rule, G-RI considers other examples in addition to the one for which the partial explanation has been derived. The reason for this is that we would like the rule learned to be generalized so that it would apply to more than the specific case under consideration. Examples for inductive learning of the

Figure 32: Explanation Missing a Rule
in the Middle

subgoal concept description are extracted by G-RI from the GBM constructed. Gemini has a knowledge base of contextual heuristics that indicate what examples to retrieve, and whether to consider them roughly as positive or negative, even if exact information about their subgoal concept membership is not readily available.

Rule induction in Gemini can be roughly broken down into two phases. In the first phase, known positive examples of the missing subgoal are retrieved from the GBM and induction from positive examples is performed. In the second phase, contextual heuristics are applied to retrieve additional examples and to further generalize the new rule.

It is possible to identify known positive examples of the missing subgoal for the first phase for two reasons:

1. For some applications, induction can be postponed until a ‘‘critical mass’’ of examples exists.
2. Examples are stored in the GBM with their *first-derived* explanations. Even if a rule was previously created for the missing subgoal, the created rule might be too specific to apply to the current case. The earlier example might need to be retrieved to re-learn the rule and make it more general.

In general, many examples may be required to generalize a new rule sufficiently. To minimize the need to wait for additional examples from the environment, Gemini makes use of examples already seen. In the absence of explicit labeling of earlier examples as positive or negative for a currently missing subgoal, G-RI makes use of explanatory information. Gemini contains a knowledge base of heuristics that indicate the type of examples to retrieve and whether those examples should be treated as positive or negative for induction of the missing concept definition. A second type of heuristic simply states sets of features that should receive particular emphasis in the new rule. The heuristics base their advice on information derived from explanations.

As an illustration of the type of heuristic that labels previously seen examples as positive or negative for the current missing concept, consider the second terrorism story given earlier in Section 3.2:

> PARIS, Feb. 4 - A bomb ripped through a crowded bookstore in the Latin Quarter tonight, wounding four people, the police said. ... The bomb that exploded tonight was planted in the basement record section of the Gibert Jeune bookstore on the Place St. Michel. The blast occurred at 7:40 P.M. with scores of customers in the store.

Say the explanation for this story, given in Figure 22, could only be partially derived because the rule to infer a crowded target was missing. (Assume that ‘‘crowded’’ did not explicitly appear as a feature, but had to be inferred.) Now say the system had previously seen an example based on the following fictional news story: ‘‘PARIS, Bastille Day - An assassination attempt was made on the President here today... The sniper ...’’ Due to its very different nature, the explanation for this earlier example would bear essentially no resemblance to the current case. Thus the system could treat it as a negative example of the concept to be learned. Furthermore, to emphasize the difference, all features indicating similarity between the examples could be removed from the new rule. This contextual heuristic can be summarized as follows:

```
‘‘If a previously seen example differs from the current
  one in its explanation leaves, subgoals, and rules,
  then:
  (1) treat it as a negative example of the currently
      missing subgoal;
  (2) eliminate all features from the new definition
      that would indicate similarity to this negative
      case.’’
```

As an example of the second type of heuristic consider the following:

```
‘‘If features under consideration already appear in
  the partial explanation, exclude them from the missing
  rule, as they are probably not required at multiple
  points in the explanation.’’
```

Contextual heuristics are intended only as a general guide for induction. As such, they are not required - or expected - to be complete. Chapter 4 describes the origin of contextual heuristics, and an algorithm for generating them. Chapter 5 describes experiments that validate their efficacy.

3.7.5. The Rule Induction Algorithm

Gemini’s rule induction algorithm is an integration of two principle ideas:

1. Learning from a conservatively selected set of positive examples provides focus on a single missing disjunct.
2. Contextual knowledge is often useful in further tailoring a missing rule.

The algorithm (shown in Figure 33) begins by finding other positive examples of the missing concept - the unproven subgoal - in memory. Recall that before induction, the current example has already been stored in the GBM. G-RI looks at earlier examples ‘‘close’’ to the current one in the GBM.[12] Of the closely related examples, those previously missing the same unproven subgoal are retrieved. Next, it performs induction over the positive examples to find the features that describe the missing subgoal. If the new rule is to be useful for completing the partial explanations of all the selected examples, its premise must refer to features shared by all of them. Sharing features is not restricted to having identical values for them. Feature values may be sufficiently similar if they have a common generalization in the static semantic memory.

```
INPUT:  example frame that could not be explained,
          corresponding partial explanation.
OUTPUT: new rule.

ALGORITHM:

1.   Select from memory a set of closely related positive examples of
     the concept to be learned;

2.   Perform ‘‘standard’’ SBL from positive examples to find the set
     of features common to all examples selected.  Essentially,
     minimal generalization from the examples is performed;

3.   Invoke contextual heuristics to retrieve additional examples
     and to explicitly focus on or discard conditions on concept
     membership.

4.   Reason forward from the rule premise and change the rule
     to reflect inferences made.
```

Figure 33: Gemini Rule Induction Algorithm

The rule produced as a result of the first part of the algorithm might require additional generalization. Contextual knowledge may provide clues as to the generalization to perform. As introduced in the previous section, a good rule of thumb might be to eliminate from the new rule premise references to all features already in the relevant partial explanations. The idea behind this is that features often do not play multiple roles in a single explanation. Gemini invokes a number of heuristics. These are the focus of Chapter 4.

The discussion of G-RI thus far has concentrated on finding a set of input features

[12] Closeness is a user-defined parameter.

that describe an unproven subgoal. However, this is not always appropriate, as the missing rule does not necessarily belong at the leaves of an explanation tree. To handle this case, G-RI attempts to reason forward from the features in the new rule premise. If subgoals are inferred - other than those already in the partial explanation trees - they replace their defining features in the rule premise. For example, say that G-RI was presented with the partial explanation in Figure 32 b. Say that the rule learned as a result was:

```
IF
    ISA (?X, fault) ∧
    HAS (?X, cond-tgt-node-responds-to-ping?, t) ∧
    HAS (?X, td-can-user-reach-another-net?, f) ∧
    HAS (?X, td-the-user-is-on-a-trn?, t)

THEN
    ASSERT (looks-like-source-prob-trn)
```

G-RI attempts to reason forward from the premise, finding that from the two features *cond-tgt-node-responds-to-ping?* and *td-can-user-reach-another-net* it can conclude *looks-like-source-prob*. The rule is modified to reflect the inference as follows:

```
IF
    ISA (?X, fault) ∧
    HAS (?X, td-the-user-is-on-a-trn?, t)
    ASSERT (looks-like-source-prob)

THEN
    ASSERT (looks-like-source-prob-trn)
```

Gemini’s rule induction algorithm differs from other approaches in a number of ways. The major difference lies with the knowledge base of contextual heuristics that guide generalization of a rule. Unlike previous work using auxiliary theories, this knowledge base:

- does not have to be complete for successful induction, and
- makes use of the structure of explanations rather than the semantics of any specific domain.

3.7.6. Summary

This section described G-RI, which learns rules missing from an incomplete domain theory. Given an input frame and a partial explanation for it, G-RI finds those features that allow the missing subgoal to be concluded. Rule learning begins with standard inductive methods, but invokes heuristics that use contextual knowledge to further guide generalization of a new rule.

3.8. Chapter Summary

This chapter has presented Gemini, an integrated learning system. Gemini makes use of SBL-like induction to learn rules missing from an EBL domain theory. SBL receives guidance in the form of contextual heuristics that make use of the explanations generated by EBL.

Three major subsystems comprise Gemini. One performs explanation generation, while the others perform two distinct types of similarity-based learning. A type of conceptual clustering is used to build a memory of input examples. This is useful to the rule induction component in that it provides it with positive examples from which to learn in the first step of induction.

Gemini makes use of a knowledge base of contextual heuristics in order to generalize new rules. These are especially useful if large numbers of examples are not available for standard induction. One goal in building Gemini was to minimize interaction required with a user. Except for the initial input to Gemini, outside guidance is virtually unnecessary. One exception is that the selection of good partial explanations is currently done by hand. This could be automated by employing a selection algorithm such as Fawcett's [Fawcett 89].

Chapter 4

Contextual Heuristics for Rule Induction

4.1. Introduction

A major focus of this research is the problem of incompleteness in explanation-based learning. Our solution introduces heuristics that guide induction of missing domain theory rules by extracting information from partial as well as previous complete explanations. The solution is based on the following claims:

1. Ultimately, inductive/empirical methods must play a major role in learning knowledge to complete a domain theory. Auxiliary theories designed for specific classes of domains may be useful, but may also fail due to incompleteness. Interaction between the system and an expert is often useful, but our goal is to minimize the need for an expert.
2. Induction can be aided by making use of the context in which learning occurs.
3. Information about explanations and EBL in general can provide focus for inductive learning of EBL domain theory rules.
4. Heuristics can be generated to extract and apply explanatory knowledge to induction.

Our solution to the problem of incompleteness is implemented in Gemini. One of Gemini's learning goals is to create rules missing from the domain theory used by EBL. In order to accomplish this, Gemini makes use of heuristics that influence the induction of domain theory rules. These heuristics guide learning by selecting specific examples from which the induction is performed. They also help Gemini focus on particular features of examples. They guide rule generalization, especially where abundant instances are not available for induction. An example of a heuristic given in Chapter 3 generalized the premise of a new rule by removing any constraints on features already in the partial explanation. Gemini's heuristics make use of the explanations derived by EBL as well as other information available in the context of explanation-based learning.

This chapter focuses on the contextual heuristics that guide induction. The next section describes earlier work on the idea of using contextual information in learning, and relates the earlier work to this research. Section 4.3 describes the origin of the contextual heuristics we have developed. It introduces a methodology for creating learning heuristics from the type of context information on which we are concentrating. Section 4.4 gives examples of the effect that such context heuristics have on concept learning as implemented in Gemini. Section 4.5 concludes the chapter with a summary.

4.2. The Use of Contextual Information in Learning

A variety of information (outside of the obvious examples and domain theories) affects the results of machine learning. We say that this information is a part of the machine learning **context**. As an example, consider an EBL system that is to reformulate the definition of a cup to include relevant observable features, as described in Chapter 2. Those features deemed to be most relevant to the definition of ‘‘cup’’ will vary, depending upon the purpose for which the cup is to be used by a performance system. The features of a cup necessary for allowing one to drink hot liquids from it, for example, are different from those that allow pencils to be held within it. The performance system’s needs are thus a part of the context in which learning occurs [Kedar-Cabelli 88].

In this section we discuss earlier approaches using context for learning. As an aid, we synthesize a hierarchy of various types of contextual information. We use placement in this hierarchy to relate our work to others’.

4.2.1. A Hierarchy of Contextual Information

Figure 34 shows a hierarchy of information that contributes to the context in which machine learning occurs. The root of the context hierarchy covers everything related to a particular machine learning system. The range of information included here is significant. It contains information that is made explicit to the system and that has obvious effects on the learning performed, such as encoded domain knowledge. It also includes information such as the state of mind of the system implementer. While the impact of this would be difficult to quantify, it would certainly affect learning.

Figure 34: A Hierarchy of Contextual Information for Machine Learning Systems

System-internal knowledge that is encoded in the the system includes inputs, outputs, built-in domain theories, and the learning algorithm. An example of external knowledge is the general purpose of the learning system — whether it is to acquire new concepts or to improve performance. If learning takes place within a particular performance system, external knowledge includes information about that as well — for instance, the performance system's efficiency constraints. The general notion of making implicit system-external knowledge explicit for learning has been explored, most notably by Keller [Keller 87] and Kedar [Kedar-Cabelli 88]. Both Keller's and Kedar's work focused on the learning task of performance improvement, rather than on the acquisition of new information. As discussed in Chapter 2, EBL is a learning method for the task of performance improvement. Keller expanded EBL by providing it with additional input specifying efficiency and correctness constraints that needed to be met by the learned generalization within the performance system. This was a first attempt at making the description of a performance system explicit to its related learning system. Kedar suggested that the concepts to be reformulated for efficiency might be reformulated in various ways depending upon the purpose they were serving within the specific performance system. Her work concentrated on extracting that purpose automatically in order to guide operationalization. In contrast to their work, we concentrate on the use of contextual information to guide the acquisition of *new* knowledge rather than the reformulation of existing knowledge.

While Keller and Kedar concentrated on bringing external information into a system, we focus on system-internal knowledge. Since our task is the induction of knowledge missing from a domain theory, we are acutely aware of the need to make use of existing information, rather than attempting to incorporate additional explicit knowledge that may be faulty. Our goal is actually two-fold, however. In addition to finding a method by which available information may be extracted and applied to induction, the method should apply *generally* to EBL systems.

The lowest level of the hierarchy in Figure 34 describes information internal to a specific machine learning system. It therefore refers to the method implemented. As an example, consider the household robot implementing SBL as described in Chapter 2. (Our focus, EBL, will be discussed in the next section.) We can abstract out general information about the method itself, shown as implementation-independent knowledge in Figure 34. Many domain- and implementation-specific decisions will be reflected in the system as well.

Although the effects of explicit domain theories in guiding learning were long accepted, Mitchell [Mitchell 80] was the first to draw attention to the various types of information that influence learning in systems traditionally viewed as knowledge-poor. Thus even in SBL systems, there exists contextual information that influences learning. Focusing on SBL, Mitchell used the term **bias** to refer to any basis for choosing one generalization over another, other than strict consistency with the observed training instances. He cited two common sources of bias in inductive learning systems:

1. the language restricts the generalizations that may be represented;

2. the procedure that searches for a single best generalization is biased.

Thus even in inductive systems that make no explicit use of auxiliary knowledge, design decisions affect learning. In the inductive variation of the robot domain, predicates were used to describe the features of input examples. The predicates did not exhaustively cover all possible features. In our example, they did not include descriptions of the state of cleanliness of cups. In order to make the learning task tractable, feature sets are often carefully selected to exclude anything deemed unnecessary for an application. As a result of their effect on learning, domain representation decisions encoded in the system itself are a part of the learning context, even for systems that do not appear to have an explicit domain theory.

While much system-internal knowledge affecting learning is implementation-specific, other information can be abstracted out to generally describe the method. For instance, in SBL one can consider the general notion that input examples are available and that they are characterized (in some representation) by some number of attributes. Our claim is that a system can make use of such general information, at least when considering EBL as a method. Our focus is on implementation-independent knowledge, as indicated in Figure 34, for the following reasons:

- It makes use of internal information already available within a given system.
- It is a basis for generalizing our approach across numerous systems implementing EBL.

4.2.2. System-Internal Contextual Knowledge in EBL

The task addressed in our research is the acquisition of knowledge missing from an EBL domain theory. Our claims, repeated from Section 4.1, are that:

1. Ultimately, inductive/empirical methods must play a major role in learning knowledge to complete a domain theory.
2. Induction can be aided by making use of the context in which learning occurs.
3. Information about explanations and EBL in general can provide focus for inductive learning of EBL domain theory rules.
4. Heuristics can be generated to extract and apply explanatory knowledge to induction.

Our solution, which introduces contextual heuristics, makes use of general information available across implementations of EBL. This section describes contextual information internal to EBL systems, highlighting general information about the method itself, which is the basis for our heuristics.

A single definitive view of EBL does not exist. (See Chapter 2.2.1. [Mitchell et al. 86] and [DeJong and Mooney 86] present two views of the method. [Ellman 89a] discusses many apparently diverse systems that may be characterized as performing EBL.) A number of key ideas, however, are shared in all views of the method, and are shown in Figure 35, which expands the hierarchy of Figure 34 for information relevant to EBL. These include the following:

- The method makes use of a domain theory. That is, a domain theory of some sort is assumed to exist. (*i-1* in Figure 35.)
- Some example description exists (*i-2*) for a concept description to be learned (*i-3*).
- The domain theory is used to generate an explanation connecting the example description to the concept description (*i-4*).
- In generating an explanation, intermediate inferences will often be made before reaching the final concept description (*i-5*).

The existence of these general characteristics of EBL differs from their actual instantiations within particular systems. For instance, both Rajamoney [Rajamoney 89] and Kedar [Kedar-Cabelli 88] implemented EBL in their own systems. Rajamoney chose processes as the representation for his domain theory, while Kedar used classification rules, as shown in Figure 35. Regardless of the choices made for the particular implementation, however, both of these, as well as others, adhere to the notion that they will have a domain theory (*i-1*) and that it be composed of units (processes, rules) that allow for intermediate conclusions (*i-5*) and that can be linked to generate explanations (*i-4*).

We have generated contextual heuristics that guide induction of rules missing from an EBL domain theory. The heuristics make use of information internally available within EBL, but they are not tailored to any particular EBL system. The generation of contextual heuristics is described in the next section.

Figure 35: A Division of EBL Knowledge

4.3. The Origin of Contextual Heuristics

The preceding section discussed various components of machine learning contexts. Machine learning systems could potentially benefit greatly by having more of this contextual knowledge made explicit to them. As mentioned earlier, both Keller and Kedar have investigated this idea for the learning task of performance improvement. Our research, on the other hand, concentrates on the learning task of acquiring new knowledge. Our claim is that EBL itself — especially in the explanations it generates — provides information useful to the construction and generalization of new rules. We concentrate on information common to a variety of EBL implementations as the basis for our creation of heuristics that guide inductive learning of missing rules. This section discusses our derivation of contextual heuristics. We give brief examples of their effect on learning. A more detailed description of their use in the context of Gemini is given in Section 4.4. A complete Gemini trace can be found in Appendix C.

4.3.1. The Goal of Contextual Heuristics

Chapter 3 described rule learning in Gemini. We will generally refer to the knowledge missing from an EBL domain theory as rules. This maintains consistency with the representation of Gemini's domain theory and with the terminology used to describe it.

Rule learning involves the determination of both a premise and conclusion. In EBL, it requires the connection of a combination of features (the premise) to a higher-level concept (the conclusion). A partial explanation given in Chapter 3 is repeated in Figure 36 to illustrate this. Rule learning may also involve the connection of intermediate concepts to higher-level concepts. As discussed in Chapter 3, intermediate concepts can be derived from more basic features. Our discussion centers on the selection of basic features for a new rule premise.

Contextual heuristics will guide a system in the selection of features to describe the premise of a new rule. As the need for a new rule is signaled by the inability to complete an explanation for a particular input example, that example provides initial

Figure 36: A Partial Explanation
with Template for its Missing Rule

constraints on features to be included in the new rule premise. The conclusion of the rule is the fact that remains to be proven in the partial explanation. All example features are candidates for the new rule premise. The task of heuristics is to focus induction on particular subsets of the example features.

4.3.2. The Intuition Behind Contextual Heuristics for Rule Induction

Consider an EBL system with a potentially incomplete domain theory. Say that the system can remember earlier examples and their derived explanations, but otherwise has no information in addition to that generally available in EBL. Even in cases where an impasse is reached in generating an explanation, there is no opportunity for the system to interact with the outside world. Even in such a restricted environment, we, as humans, could apply various rules of thumb in selecting the example features relevant to a missing rule. We can divide those heuristics into three general categories:

i. Look at the role a feature plays within the current partial explanation. For instance, say that a feature is already referenced in the partial explanation generated for the current input. It might be reasonable to assume that the feature won't play multiple roles in the same explanation and so can be discarded from the premise of the missing rule.

ii. Look at the role a feature plays in another explanation. For example, say a feature (with its corresponding value) of the current (partially explained) example was referenced in the explanation of a previously seen example. Also, say that the earlier example differed greatly from the current one both in the example itself and in its derived explanation. One might again eliminate the feature from the new rule premise due to its salience in a very different situation.

iii. Look at the relationship between the current example and a previous one. Suppose an earlier example was an instance of the same goal concept as the current one but that this was proven in a very different way. One might assume that the difference between the disjunctive proofs of the single goal should be maintained. The old example could be treated as a negative example for induction of the currently missing rule.

Such general heuristics are useful for a number of reasons:

1. The third class described allows for the selection of positive and negative examples for induction. It can make use of earlier examples, thus not requiring that a system wait for additional input from a teacher. It uses explanatory information to treat examples as positive or negative without requiring that they be labeled explicitly by an outside source.

2. The first and second classes of heuristics provide cues for additional generalization of a new rule beyond simple induction. This is useful when large numbers of examples are not available.
3. Heuristics make use of generally available information. They do not require an auxiliary domain theory (beyond the heuristics themselves), and they do not require additional interaction with the environment. Furthermore, these heuristics need not be created by a clever system implementer. They can be generated automatically from a list of general information available to EBL.

4.3.3. The Derivation of Contextual Heuristics from EBL Attributes

General attributes of EBL provide a ready source of information that may be useful in guiding the acquisition of new rules. They are a basis for heuristics that guide induction. Heuristics explicitly focus attention toward or away from individual features based upon the role they fill in the current example as well as in others. In the preceding section three heuristics were presented that one might find useful for the induction of missing EBL rules. These heuristics made use of explanatory information and a history of examples:

1. to label instances as positive or negative for induction;
2. to otherwise generalize the premise of a missing rule.

Consider the following example of a heuristic that treats an old instance as a strongly negative one for induction:

```
IF the current example (with its corresponding explanation)
     has the same goal as an earlier example
   AND
     refers to different intermediate concepts in its
     explanation than the earlier one
THEN eliminate from the new rule all features covered
     by the features in the earlier example.
```

(An example using this heuristic was given in Chapter 3. In that example a Bastille Day assassination attempt was treated as a negative example for the induction of a definition of a crowded target.) The heuristic is essentially an if-then rule, where

- the premise indicates the type of examples to consider;
- the conclusion states what to do with various features being considered for the new rule.

The premise of a heuristic indicates the conditions under which an earlier example is to be retrieved for induction. It defines a relationship between the current unproven example and some earlier example. A relationship can be specified by indicating the similarity or difference between the examples with respect to generally available information. This information includes the following:

1. the features of the examples;
2. the explanation goals for the examples;
3. the subgoals proven in generating the (possibly partial) explanations;
4. the unproven subgoals, if there are any;
5. the features referenced in the leaves of the explanation trees;
6. the rules used in the explanation trees.

The first two are available from the input to EBL; the second through sixth are available from the output generated from EBL. Thus they will be available regardless of the EBL implementation. In the sample heuristic given above, only two of these attributes appear in the premise: the goal and the subgoals. In general, any subset of these attributes can be ignored in specifying a relationship in a heuristic condition.

The conclusion of a heuristic indicates the manner in which features are to be treated when determining their relevance to the missing concept definition. A feature (or set of features) may be retained or eliminated based upon its presence in the retrieved example. If the retrieved example is believed to be positively related to the current missing concept definition, features appearing in the retrieved example are retained; otherwise they are eliminated. A heuristic might refer to input example features in general, or it may refer to some subset of features, such as:

1. those from which the goal has been inferred;
2. those from which the subgoals have been inferred;
3. those appearing in the leaves of the explanation tree;
4. those appearing in certain rules used in the explanation.

In Gemini these are all identical, but they need not always be.

In general, a contextual heuristic is an if-then rule that can be generated by filling in the templates shown in Figure 37 A. Figure 37 B shows a heuristic in which no condition is specified. It simply issues a directive about the way to treat features of a certain type. This corresponds to the class of heuristics described in the previous section in which a feature is treated a certain way based upon its role in a particular example. For instance, from template B the following contextual heuristic could be generated:

```
eliminate from the new rule all features covered
by the leaves in the current example.
```

Note that template B refers only to features covered by the current example as opposed to features of the current example covered by the previous one. A allows for the indication of relationships among examples. B allows for the indication of relevance within an example.

From the templates in Figure 37 a huge number of heuristics could be generated. Appendix A describes a subset of them (94 heuristics) in detail. Not all of these will be useful. In order to verify the efficacy of a subset of contextual heuristics, we selected ten for testing. The results of our experiments are discussed in Chapter 5. Of the ten heuristics selected, some were chosen because it was believed they would prove useful. Others were selected to verify that they would fail. Yet a third subset was chosen because its performance could not be easily predicted before the experiments. The following section describes the heuristics tested.

4.3.4. A Discussion of Selected Heuristics

This section describes ten heuristics selected for testing in Gemini. The heuristics are written using the notation of the templates in Figure 37.

Test Heuristic 0:

```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

This is a modification of heuristic #12 from Appendix A. It looks for other positive examples of the missing concept (in fact, those most similar to the current instance), and then performs least generalization from those examples (i.e., simple-minded SBL). The

```
A.  IF

      current example x is SAME AS | DIFF FROM | DON’T CARE
      past example y in attribute 1

      AND

      .
      .
      .

      current example x is SAME AS | DIFF FROM | DON’T CARE
      past example y in attribute 6

    THEN

      KEEP | ELIM | IGNORE features in rule that are the same as those
      covered by attribute 1 in example y;

      .
      .
      .

      KEEP | ELIM | IGNORE features in rule that are the same as those
      covered by attribute 6 in example y.

{attribute 1 = explanation features; attribute 2 = goal concept;
 attribute 3 = subgoals;             attribute 4 = leaves;
 attribute 5 = unproven subgoals;    attribute 6 = rules used}

B.  IF

      no condition specified  /*all don’t cares*/

    THEN

      KEEP | ELIM | IGNORE features in rule covered by attribute 1
      in current example x;

      .
      .
      .

      KEEP | ELIM | IGNORE features in rule covered by attribute 6
      in current example x.
```

Figure 37: Templates for Generation of Heuristics

condition of this heuristic most concisely describes the situation where all features of the examples are the same. The similar rules imply that the explanations (and thus the goals, subgoals, and leaves) are similar.

Test Heuristic 1:

```
IF
```

```
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Here the similarity of the subgoals implies that the difference between the two explanations lies *only* in the final step to the conclusion. This is, in fact, a reasonably common occurrence in our diagnostic domains. It occurs in the situation where two problems appear similar until attributes are finally tested that distinguish between them. As the difference between the examples is assumed to be isolated to just one position, the examples are taken to be positive examples of the missing concept.

Test Heuristic 2:

```
IF
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

The conditions here are the same as those in the previous example, but they are taken to imply that the two examples are sufficiently different that they are to be covered by separate disjuncts of the definition of the missing concept. We tested this heuristic believing that it would fail while Heuristic 1 would work. This hypothesis was shown to be correct.

Test Heuristic 3:

```
IF
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

This heuristic is very similar to Heuristic 1. In this case, however, the condition of the heuristic requires that the features referenced in the leaves of the explanations be the same. This occurs in our diagnosis domains when a single attribute's *value* is used to distinguish between two input examples.

Test Heuristic 4:

```
IF
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

This heuristic is the opposite of Heuristic 3. Though the examples appear quite similar, it is possible that their leaves are actually somewhat different. As a result of Gemini's implementation of heuristics, similarity of leaves in the condition indicates that similar leaf *names* are used. It does not imply that their values are the same. Thus the two examples are treated as sufficiently different to belong to separate disjuncts of the missing concept definition. Heuristics 3 and 4 were selected for testing as cases where we could not truly hypothesize their efficacy, though our suspicion was that 3 would be successful while 4 would fail.

Test Heuristic 5:

```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Although the frame description of the retrieved example y is similar to x's, and though they are members of the same goal concept, the features important for classifying each as a member of that concept are quite different (as indicated by the difference in their respective explanations). In order to follow this thread, this heuristic concentrates on the features of x that are different from those in y. This is heuristic #18 from Appendix A, and is a version of the ‘‘disjuncts are different’’ rule from Chapter 3.

Test Heuristic 6:

```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y;
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Although the retrieved example y differs from the current example x, it is also a positive example of the concept to be learned — i.e., the missing subgoal. Furthermore, y is also quite similar to x in its input representation. Thus, it is possible that the two examples should be covered by a single definition for the missing subgoal. As a result, their shared features are noted in the new rule. Additionally, features from y's explanation are deleted from the rule as they would probably have been mentioned in x's explanation if they were important there. Note that there is sufficient difference between x and y that the assumptions of this heuristic could be quite wrong. This heuristic is #26 from Appendix A. It was actually expected to do reasonably well in the network faults domain, which is what our tests showed.

Test Heuristic 7:

```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Here the retrieved example y, although appearing similar to the current example x in its input representation, is really very different in its stated goal concept membership as well as in its explanation. It is therefore reasonable to treat it as a negative example of the missing subgoal to be learned. As a result, attention is drawn to the features of x that differ from those in y. This is heuristic #27 from Appendix A.

Test Heuristic 8:

```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
```

```
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Though the retrieved example y looks quite different from the current example x, it is a positive example of x's missing subgoal. It is taken to be an example that should be covered by a different disjunct of that concept than x is. In order to keep that distinction visible in the definition to complete x, features relating x and y are eliminated from the definition. A somewhat conservative approach is taken, however, eliminating only a subset of their shared features. Note that the conditions of this heuristic are a generalization of those in #17 and #48 of Appendix A. This is heuristic #79 from the appendix.

Test Heuristic 9:

```
  ELIM features in rule that are covered by LEAVES in current
     example x.
```

This heuristic was introduced earlier. It assumes that features of an input example will generally not be referred to multiple times within a single explanation of that example. As a result, all those features already referenced may safely be eliminated from the rule being constructed to complete the explanation. This is heuristic #94 from Appendix A.

To summarize, heuristics were selected for testing on the following basis: Heuristics 0, 1, 5, 7, 8, 9 were predicted to perform well in testing. Heuristic 2 was expected to perform poorly. The predicted behavior of Heuristics 3, 4, and 6 was somewhat unclear. The following section traces through the application of two heuristics to a Gemini example. Chapter 5 discusses our experimental results.

4.4. Using Heuristics for Rule Learning in Gemini

The previous section on the derivation of our contextual heuristics gave examples of some of them. This section traces a typical rule learning session in Gemini in order to illustrate contextual heuristic use in the system. The trace is taken from the domain of manpack radio fault diagnosis. The domain was chosen to illustrate rule learning due to its simplicity. In this domain, the radio diagnosed is a communication radio used by the military. It is a pack radio, to be carried by an individual. As in the network fault domain, knowledge is encoded on a level that enables isolation of the fault to a particular major radio component, but does not allow deeper analysis of the problem. The rule base for this domain is a direct translation of troubleshooting procedures listed in the manual for the radio [Radio Manual 86].

As discussed in Chapter 3, input enters Gemini through its EBL component, G-EBL. The input is standard EBL input of a goal concept and an example of that concept. The goal of G-EBL is to construct an explanation describing why the example is an instance of the goal concept. Say the input is the frame given in Figure 38 A. Input frames describing fault instances in this domain are single-level frames containing binary valued features. The features describe two types of information: (1) observed symptoms of a problem; (2) results of tests performed on the radio. The value of a feature may be unknown. An abridged version of the input frame excluding unspecified features is shown in Figure 38 B.

In the manpack radio domain, Gemini's task is to derive an explanation that links a fault to its diagnosis. The goal concept for the input fault shown in Figure 38 is *rt-bad*. Figure 39 gives the full explanation that would be derived by Gemini for the input example.

Now say that the following rule allowing Gemini to infer *rt-or-KY57-or-comsec-bad-prefer-KY57* was missing from the rule base:

```
IF
    HAS (?X, S-KY57-alarm-tone-or-beep-not-heard-at-rt, T)
```

Figure 38: An Input Frame to Gemini

```
THEN
    ASSERT (rt-or-KY57-or-comsec-bad-prefer-KY57).
```

Gemini would be able to generate only the partial explanation in Figure 40. Gemini's rule induction component, G-RI, is now required to find an appropriate rule to complete the explanation. Its goal is to find a rule that not only fits the current example, but that will be applicable to others as well.

As other positive examples will be needed for the induction of a new rule, Gemini

Figure 39: Gemini's Explanation of the Input Example

first places the example with its corresponding explanation in memory. The memory is built by Gemini's conceptual clustering algorithm. G-CC considers the similarity of both input features and explanations in deciding whether inputs are related. Say the current input example is directly related to only two other examples, shown in Figure 41. Say that the original explanations generated for these examples were identical to the current partial explanation. Say they were completed with rules that were too specific to apply in the current case.

Assume that there are also a number of other non-related examples in memory. One of these, which will be used later, is shown in Figure 42 with its corresponding explanation.

Figure 40: Partial Explanation Generated when a Rule is Missing

Once the input example with its corresponding explanation has been placed in memory, G-RI is ready to begin the creation of the missing rule. As discussed in Chapter 3, G-RI treats rule induction as concept learning from positive examples. The conclusion of the missing rule is the unexplained subgoal — the concept to be learned. G-RI must induce the rule premise — or input features — that is relevant to making the inference to the subgoal, as illustrated in Figure 43. In this example, G-RI must find the features of the fault that imply the partial diagnosis *rt-or-KY57-or-comsec-bad-prefer-KY57*.

Figure 41: Closely Related Examples
Retrieved for Induction

G-RI's first step is to find a core set of potentially relevant features by treating the rule induction problem as concept learning from positive examples. Rather than requiring that a user be present to supply it with additional examples, Gemini searches its input memory for those. It is possible that the missing concept definition is disjunctive. In order to focus on the one disjunct required for the current input, the examples retrieved:

1. must have been missing the inference to the same subgoal as in the current case, and
2. must be similar to the current example with respect to its features, and
3. must be similar to the current example with respect to its explanation.

As such examples are generally siblings (or at least closely related) in the memory, ‘‘search’’ for other positive examples is essentially a look-up operation. In this case, G-RI retrieves the two related examples Figure 41.

The retrieved examples are compared with the current one and a maximally specific generalization is found that classifies the three. If the new rule is to cover all three examples it must refer to features common to them. The basis of the

Figure 42: An Additional Example in Memory

generalization algorithm is the pairwise matching algorithm used in G-CC and given in Chapter 3. When multiple instances are to be matched, first one retrieved example is

Figure 43: Missing Rule to be Completed by Gemini

compared against the current one. The remainder are incrementally compared against the resulting generalization. Figure 44 gives the result of comparing the input frame with the retrieved examples.

Figure 44: The Result of Comparing Positive Instances of the Missing Subgoal

If Gemini were to stop now, the new rule taken from the generalized frame would be the one shown in Figure 45.[13] The rule in Figure 45 is obviously far more specific than the missing rule. In fact, given the instances retrieved, all features are still considered potentially relevant for the premise of the new rule. This is a natural result of the fact that Gemini had only a couple of additional positive examples from which to perform induction. This will often be the case in a (primarily) autonomous learner that cannot depend upon a user for additional input. In addition, diagnostic domains such as this one suffer from what we have termed the ‘‘healthy features effect’’. This means that an inductive learner using positive examples has no way to distinguish shared but

[13]Translation of frames to rules is implemented in Gemini.

healthy features from those that truly identify the fault. G-RI's goal is to further generalize the rule so that it will be more widely applicable. In order to accomplish this, Gemini makes use of heuristics that take advantage of information already available to the system. The heuristics make use of the current explanation as well as other examples and their explanations. The system could conceivably wait for more positive examples, but they might not be available on a timely basis. The system therefore makes use of the information it already has. This trace will show the use of two heuristics.

Figure 45: The Rule Created from Existing Positive Examples

The Application of Test Heuristic 5:

```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

This is the ‘‘disjuncts are different’’ heuristic described earlier. Essentially, it says that if an identical goal is explained in different ways, the difference should be maintained throughout the explanations. Searching through the memory for an opportunity to apply this heuristic, Gemini finds the fault shown in Figure 42. After applying the heuristic action, the new rule is as shown in Figure 46. Treating the retrieved example as a roughly negative one for induction of the missing concept, Gemini was able to significantly generalize the new rule. Another heuristic is now applied to further generalize the new rule.

Figure 46: The Rule After Applying One Heuristic

The Application of Test Heuristic 9:

```
ELIM features in rule that are covered by LEAVES in current
   example x.
```

This heuristic has already been discussed a number of times. Essentially, it states that any features already playing a role in the partial explanation will probably not be needed an additional time in the missing piece. After applying this heuristic, Gemini generates the rule in Figure 47, which is identical to the one missing from the domain theory.

The examples in this section have been designed to illustrate the utility of

Figure 47: The Rule After Applying a Second Heuristic

contextual heuristics. Heuristics are not guaranteed to work, however. Chapter 5 gives an evaluation of ten contextual heuristics, showing that they are effective in general, and characterizing their performance in different domains.

4.5. Summary

A major claim of this research is that contextual information is useful for finding the knowledge missing from a domain theory. A large range of information may be termed contextual for machine learning in that it affects the outcome of machine learning, either explicitly or implicitly. Past approaches to using contextual information have concentrated on making implicit goals explicit to the machine learning system. The learning task addressed in these cases was performance improvement. In contrast, this research concentrates on tapping into knowledge that is already part of a system, and using that to guide the acquisition of new knowledge.

As the primary problem addressed by this work is that of completing EBL domain theories, we concentrate on information that is generally available within the inputs, outputs, and domain theory of EBL itself. This chapter has given an itemization of general attributes of EBL systems — potential sources of information that are not tied to a particular implementation. The attributes are a basis from which heuristics are derived for guiding the acquisition of new rules. Although not guaranteed to work, many heuristics do result in correct and general new rules. (Chapter 5 gives a detailed evaluation of ten heuristics as tested in Gemini.) Furthermore, they are an improvement over previous approaches to the problem of domain theory incompleteness for the following reasons:

1. They make use of existing information rather than relying on an auxiliary theory that may itself be incomplete. The knowledge base of heuristics need not be exhaustive to be useful.
2. They do not require additional input from or interaction with a user.
3. They guide an inductive system in performing generalization that would otherwise require more examples. If many examples are available for induction, then straightforward inductive techniques are a safe approach for acquiring missing information. Abundant examples are not always available, however. Furthermore, in an autonomous system that receives inputs incrementally it may be detrimental to wait for a larger number of inputs, even if they exist.

This chapter has described contextual heuristics derived from EBL attributes, giving an example of their use in Gemini. The following chapter gives an experimental evaluation of contextual heuristics.

Chapter 5

Evaluation of Contextual Heuristics

5.1. Introduction

Induction is ultimately necessary in creating new knowledge to be incorporated into an incomplete domain theory. Though inductive algorithms implicitly have knowledge encoded into them, they are less dependent than deductive systems upon potentially imperfect auxiliary knowledge. Knowledge is valuable in guiding learning, however, when it exists. Chapter 4 described domain-independent heuristics that make use of explanatory knowledge in order to guide the induction of information missing from an EBL domain theory. This chapter describes results of test runs that support the claim that these heuristics are effective in guiding the induction of missing knowledge.

We show that:

- Heuristics are (empirically) effective in guiding induction of correct rules and can do so with fewer examples than would be required by induction without such heuristics.
- Induction can proceed without the need for labeling examples by an external source, since heuristics determine whether examples are to be treated as negative or positive instances for induction.
- Heuristics complement each other and can be used effectively in combination.
- The heuristics are domain-independent, and rely only on general types of information available in any explanation-based learning system.
- Though they constitute a type of auxiliary knowledge, heuristics are not assumed to be perfect and are therefore not subject to the same problems as auxiliary domain theories.

This chapter presents experiments and data to verify these claims. In addition, the experiments give a characterization of a subset of heuristics including the following information:

- Partial lists of heuristics that perform well in general and those whose performance varies across domains.
- A characterization of the quantity of information they make use of within a memory of typical input instances.
- A characterization of the types of attributes they help to generalize in a learned rule. Different heuristics focus on different features in generalizing.

The discussion of our results begins with a description of the general experimental methodology in Section 5.2. Included in the description are the general methodology itself, an explicit itemization of assumptions, an introduction to the domains tested, an itemization of heuristics tested, and our criterion for evaluating system output. Section 5.3 gives an overview of the results that are then described in detail (experiment by experiment) in Section 5.4. Section 5.5 summarizes the results.

5.2. Methodology

This section describes the methodology followed in our test runs. Four domains were tested:

1. network fault diagnosis
2. diagnosis of a military manpack radio
3. diagnosis of a military vehicular radio
4. analysis of terrorist event news stories.

The basic test method involves removing rules from a domain theory and then determining whether inductive learning using each of a set of heuristics is able to re-learn the rules. We describe the design of the test runs in the remainder of this section. We follow that with an introduction to the results. A uniform experimental method was followed in each of the four domains, as described here. Where there are slight departures from this method, they are made explicit in the discussion of the individual runs in Section 5.4.

5.2.1. Experiment Design

Each test was performed following a single basic methodology, summarized in Figure 48.

Begin with a domain theory to be treated as complete and correct.

Enter instances to the learning system for explanation and storage.

Eliminate a rule to determine whether it can be recreated by the learning system. Multiple rules may be eliminated, but only one rule will be learned at a time.

For each input instance, generate an explanation again. Simulate the generation of partial explanations to guarantee goodness. If the rule(s) removed from the domain theory appear in any of the explanations previously derived, remove them from the explanations. Also remove any subtrees below them in the explanation trees.

To evaluate the performance of a given heuristic *h*:

1. **Given an instance *n* with a partial explanation (called the *trigger instance*), retrieve from the system's memory other instances that meet the conditions for applying *h*. Retrieve variable-sized sets of instances to determine the effects of having a variable number of instances for induction. Repeat this for other trigger instances.**
2. **Apply the actions of heuristic *h* to each of the variable-sized sets of retrieved instances for each trigger.**
3. **Evaluate each learned rule based upon its similarity to the original rule.**
4. **Repeat for each heuristic.**

Figure 48: Experiment Methodology

In each test, we began with a domain theory assumed to be complete, correct, and tractable. As stated previously, such an assumption is unrealistic in many real-world applications. Rather than imposing such standards on the test domains, we used the domains as they were originally given, but treated them as essentially perfect. This is reasonable as:

1. It allowed us to study the performance of our theories in a more realistic setting.
2. The original given domain theory was probably at least reasonable given that its source was an expert in three of our four domains.

For each domain, we next input a set of instances to Gemini. Recall from Chapter 3 that each input to Gemini consists of a pair of a concept name and an instance of that concept. For each input, Gemini creates an explanation describing why the instance is, in fact, an example of the concept. For each input in each of our domains, Gemini successfully generated an explanation. That is, the rule bases were sufficiently complete for generating the necessary explanations.

To evaluate the performance of our contextual heuristics, we removed rules from the rule base. The focus of each test run was to determine whether Gemini could recreate the rule(s) removed, as discussed in Section 5.2.2. Gemini is able to handle multiple missing rules. Since it learns a single rule at a time, however, we generally removed a single rule at a time for testing.

Next, explanations for each of the instances were generated again. Some of these were partial because they required the rule(s) removed from the domain theory. To generate partial explanations, we deleted the missing rules from the previously generated complete explanations. Ideally, the partial explanations would be generated automatically. However, generating good partial explanations is an extremely difficult problem, as discussed in Chapter 3. In order to best evaluate the heuristics themselves, we assumed that the explanation generation (including that of partial explanations) was working as well as possible. Thus we started with the known good explanations, and modified them to reflect the gaps in the domain theory. In addition to removing the missing rules, we deleted any parts of the explanation tree that would fall below them. If explanations are generated backwards, reasoning from the root (goal concept) to the instance, the absence of the removed rule would stop explanation generation below it.

In order to apply heuristics, Gemini stores instances and their associated explanatory information in memory. The instances are later retrieved for induction and the heuristics indicate how information in them is to be used.

Next we evaluated the performance of each heuristic. Gemini was given an input example for which only a partial explanation was generated. To evaluate the performance of each heuristic, Gemini invoked its rule induction component, but was allowed to use only the heuristic being tested. During the course of normal operation, a system such as Gemini would run until it found that it was missing necessary information. When an input arrives for which an explanation cannot be generated, those features appropriately implying the missing concept must be selected. The input acts as a ‘‘trigger’’ to set in motion the heuristics that find and then infer from past inputs. To mimic this process in a controlled experiment, we built a memory of the input instances given to Gemini earlier. Then, for each of a set of *trigger instances* (a subset of the input instances), Gemini accessed its memory to retrieve instances to which a given heuristic could be applied. Gemini then applied the actions of the heuristic. Recall that a heuristic is an if-then rule that specifies conditions on applicability and an action to be followed for rule induction. When a heuristic is applicable with respect to a previously seen instance, we say that the instance is *retrieved* for induction.

Depending upon the system’s history, the heuristic might retrieve any number of previous instances. We studied what happens to the induction with different numbers of instances. We allowed a heuristic to retrieve the maximum number of instances to which the heuristic applied. We then varied the number of instances actually used in the induction. We did this for each of a number of randomly selected trigger instances, in order to get a more ‘‘average’’ picture of a heuristic’s performance. For example, in the domain of network fault diagnosis, when a rule named RULE1 was deleted from the rule base, we selected ten instances to act as triggers. For each of the ten trigger instances, Gemini then used each heuristic to retrieve other instances from its memory. It retrieved instances in varying numbers. For one heuristic, it retrieved anywhere from 0 to 60 instances, which we studied in increments of 10.

Next, the actions of the heuristic being evaluated were applied to each set of instances retrieved for induction. There were many sets of instances, because each trigger instance caused the retrieval of a number of relevant instances which were studied in variable sized sets. To evaluate a learned rule, we compared it with the rule that was originally removed from the domain theory.

We repeated this general method in each domain, for each heuristic, for each of two or three deleted rules. Rules were deleted from various positions in their respective generated explanations. Some referred only to input features in their left-hand-sides, and would therefore only fall at the leaves of any given explanation. Others referred both to input features and to intermediate subgoals, and would therefore fall in the middle of generated explanations.

5.2.2. Evaluation of Learned Rules

Rules learned by Gemini using each of the heuristics were evaluated based upon their similarity to the rules deleted. The primary justification for this is the fact that three of our four domains were taken from outside sources (i.e., were not constructed by us) and were designed by ‘‘experts’’ in their fields. We also evaluated one set of learned rules based on their performance accuracy. (See Section 5.6.) We found that the goodness of a learned rule with respect to performance mirrored its goodness with respect to similarity to the deleted rule. Thus evaluating learned rules according to their closeness to expert rules reflects both their performance accuracy and their conceptual integrity.

In presenting the results of our test runs we give three important pieces of information. For each run we provide:

1. The number of additional conjuncts in the new left-hand-side. We give both the number of features mentioned that were not in the original rule and the number of intermediate concepts that were not in the original rule. This gives a general indication of new rule’s degree of *over-specificity*.
2. The number of conjuncts missing from the new left-hand-side. We give both the number instance features not mentioned in the new rule and the number of intermediate concepts that appeared in the original rule but that didn’t here. This gives a general picture of the new rule’s degree of *over-generality*.
3. The actual features and subgoals appearing in the learned rule. As will become apparent, different heuristics focus on different subsets of features.

As the amount of data is extremely large, it is discussed from two different perspectives, focusing first on the individual domains and then focusing on the heuristics themselves. We describe the experiments performed in each of the domains, discussing the ways in which individual rules were learned using each of the heuristics. Later we discuss the heuristics, describing how they performed overall.

5.2.3. Domains

Tests to characterize the performance of our heuristics were run in four domains: network fault diagnosis, manpack radio fault diagnosis, vehicular radio fault diagnosis, and terrorist event news story analysis. We give brief summaries of the domains in the discussions of the experiments. More details on each of these domains can be found in Appendix B, including brief domain descriptions, the actual rule bases, and other relevant domain knowledge encoded in Gemini.

5.2.4. Heuristics Tested

Chapter 4 discussed heuristics for rule induction. There we introduced both the origin of our heuristics and a mechanism for itemizing an exhaustive set of them. As addressed in that chapter, the total number of possible heuristics is too large to test exhaustively (without virtually unlimited resources). For purposes of testing, we selected 10 heuristics as itemized in Chapter 4. The 10 heuristics are enumerated again in Figure 49, Figure 50, and Figure 51. When selecting heuristics for testing, we attempted to satisfy the following criteria:

- We tried to choose heuristics with a wide range of applicability. That is, we selected heuristics whose conditions on applicability would cover a wide range of input instances.
- We tried to select some heuristics that we hypothesized would work well; some that we hypothesized would fail; and others about which we could not make a hypothesis.

In Chapter 4, we gave brief descriptions of the 10 heuristics, including our hypotheses about their performance. In Figure 49, Figure 50, and Figure 51 we repeat the performance hypotheses. Appendix A gives a listing of a much larger set of generated heuristics, with descriptions of each. There we also give a brief guide to easy reading of the heuristics.

```
Test Heuristic 0: ‘‘Most conservative least generalization
                    of positive instances.’’
Hypothesis: success

IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.


Test Heuristic 1: ‘‘Least generalization from liberal instances.’’
Hypothesis: success

IF
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.


Test Heuristic 2: ‘‘Different goals mean no relationship - even
                    if some things shared.’’
Hypothesis: failure

IF
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Figure 49: Test Heuristics 0 - 2

5.2.5. Assumptions

Assumptions made in the test runs to characterize our learning heuristics were mentioned in Section 5.2.1. We repeat them here for emphasis:

- All domain theories are assumed to be ‘‘reasonable’’. Domain theories for three of the four domains used for testing were obtained from expert sources. Though there is no guarantee that these domains theories are complete, correct, and tractable, we treat them as such. This allows us to test our heuristics in a realistic setting.
- Original rules are assumed to be good, as they were expert-generated. Thus recreating them is assumed to be a reasonable goal for the learning system.

```
Test Heuristic 3: ''Conservative least generalization.''
Hypothesis: success

IF
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.


Test Heuristic 4: ''Different goals mean no relationship - no
                    matter how similar things look.''
Hypothesis: failure

IF
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.


Test Heuristic 5: ''Disjuncts are different.''
Hypothesis: success

IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Figure 50: Test Heuristics 3 - 5

- A good partial explanation of an instance for which a rule is needed is assumed to be the same as the complete explanation missing that rule and all that would be necessary to infer it.

```
Test Heuristic 6: ''Very liberal generalization
                    from positive instances.''
Hypothesis: success in network faults; no prediction otherwise.

IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y;
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.


Test Heuristic 7: ''Features irrelevant - explanation's
                    the determining factor.''
Hypothesis: success

IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.


Test Heuristic 8: ''Disjuncts are different - but conservatively.''
Hypothesis: success

IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.


Test Heuristic 9: ''Features used at most once.''
Hypothesis: success

  ELIM features in rule that are covered by LEAVES in current
     example x.
```

Figure 51: Test Heuristics 6 - 9

5.3. Introduction to the Results

The highlights of our results are:

- Heuristics 0, 1, and 3 (the *conservative least generalization* heuristics) are effective in labeling instances as positive for induction. They never resulted in over-generalization, though they resulted in some of the over-specialization common when negative instances are not considered. Heuristic 6 (*liberal generalization from positive instances*) also worked quite well, though it was better at generalizing. Heuristic 8, a conservative version of the *disjuncts are different* heuristic, was successful as well. These outcomes matched our performance predictions.
- Heuristics 5 and 7, which treat instances as strongly negative, appear at first to have erratic behavior. However, they perform extremely well in generalization when only very few instances are retrieved. This is consistently true regardless of the actual instances retrieved. We had predicted successful results with these heuristics, though we had not anticipated requiring a restriction on the number of instances used for induction.
- Heuristic 9, which assumes that features will appear at most once per explanation worked appropriately in all our domains. Though it is limited by the size of the partial explanations, it is a useful heuristic to apply as it does generalize and requires virtually no resources to carry out.
- Heuristics 2 and 4 treat retrieved instances as negative instances for induction even though they are similar to the trigger instance. As expected, these performed poorly in terms of our evaluation criterion.

A combination using one of {0, 1, 3, 6, 8}, one of {5, 7}, and heuristic 9 are an effective combination for induction. These heuristics make use of explanatory knowledge to label instances as positive or negative for induction, and indicate the degree to which that is the case. This is done without the intervention of an outside source. Instances labeled as strongly negative allow numerous features to be generalized away from a concept definition that would otherwise require more instances.

5.4. Experiments

This section discusses experiments run in each of the four domains described earlier. Due to the large quantity of data and the variety of perspectives from which it can be viewed, the data are described from two selected perspectives. First, experiments are organized around domain, describing the relative performance of heuristics in a single domain at a time. In Section 5.5 results are described from the perspective of heuristics, discussing how each performed across domains.

5.4.1. Network Fault Diagnosis

Figure 52: Encoding of Network Fault Attributes

This domain, described earlier, involves the diagnosis of an ethernet/token ring network. The knowledge used by the performance system enables isolation of a fault to a particular segment of a network, but does not allow deeper analysis of a problem.

Input frames describing fault instances in this domain are single-level frames containing 27 binary-valued attributes. An input description may be incomplete, i.e., some of the attribute values may be unknown. The complete set of input frame attributes is given in Figure 52. Each attribute is preceded by a number. These numbers will be used later in the section when abbreviations of the attributes are necessary.

The rule base for this domain contains 56 rules. Rules were translated into Gemini's representation from a prototype knowledge base that was extracted from experts maintaining the CMU campus computer network [Eshelman 88]. We deleted the three rules in Figure 53 to perform test runs characterizing the behavior of learning heuristics in this domain.[14] The following sections describe the runs we performed for each of these three rules.

Figure 53: Rules Deleted for Testing in Network Fault Diagnosis

[14]The representation used here varies slightly from the actual representation for clarity.

5.4.1.1. Experiments for Missing Rule #1

This section describes test runs where the rule labeled RULE1 was removed from the domain theory for re-learning. The left-hand-side of the rule refers only to input attributes. Thus it would always appear at the bottom of an explanation tree. Sample explanations are given in Figure 54, the first of which is a typical explanation using the rule, and the second of which is the partial explanation derived when the rule is missing. Both explanations were shown earlier in Chapter 3.

In performing the runs, Gemini has access to a memory of instances which the system has already analyzed. Heuristics make use of the memory by retrieving specific instances that meet their criteria for applying a particular action. The memory of instances used in the test runs here contained 90 instances. Eighty were partial as they required RULE1 in order to complete them. Of these 80, 10 instances were selected at random as trigger frames for induction. Ten were selected to give a picture of the heuristics' general performance. The frame designated as the trigger frame was removed from memory for the duration of the time it served as trigger.

For each trigger frame, a test set was run with *each* heuristic to determine whether the removed rule could be recreated. Because in reality the memory will contain a varying number of instances over time, varying numbers of instances were retrieved, ranging from 0 to the maximum possible given the memory. We varied the number of instances retrieved for two reasons:

1. To more accurately simulate learning during a normal course of operation. During normal operation the memory is built over time. It does not remain static.
2. To determine the ''best'' number of instances for any given heuristic to use. We did find that certain heuristics worked better with larger sets of instances, while others performed well with small sets. Retrieving additional instances with those heuristics provided no additional information.

The number of instances retrieved for induction followed the pattern: 0, 1, 5, 10, and then continued in increments of 10.

Figure 55 summarizes the results of applying each of our test heuristics to learning

Figure 54: Explanations using Network Fault Domain
RULE1

RULE1. A row in the table describes results for a single heuristic. Recall that each heuristic was tested for this rule using 10 different trigger instances. Each row in the table gives the results for only one representative trigger instance. Each column indicates the number of instances retrieved from memory by the heuristic and to which the actions of the heuristic were applied. Two primary values are given for each table entry:

- The first value is the number of extraneous features in the learned definition of the missing concept. This generally indicates that the rule is too *specialized*. That is, extraneous features in the left-hand-side of the learned rule impose additional constraints on the applicability of the learned rule.
- The second value gives the number of features missing from the definition. Often, this indicates *over-generalization*, because conditions on the rule's applicability are missing. The task is to create new rules that are general without being over-generalized.

The italicized values give the range of numbers of extraneous features across all 10 triggers. (Recall that only one representative trigger is given to show the results for each heuristic.) The final column of the table lists the features mentioned in the learned rule that were using the maximum number of instances in that row. That is the most generalized rule that can be learned using the heuristic. Thus it gives an indication of whether the heuristic tends to over-generalize or over-specialize, and the features on which it tends to focus. We list only attributes, rather than both their names and values, because in this case the values are always the values of the attributes in the trigger instance. Recall that the trigger instance's features are the basis from which selections are made for the new rule's lhs. Features are listed by their number codes, given earlier in Figure 52. As shown in Figure 56, the features that should be selected for this rule are 1, 6, and 12 (source can't connect, target looks fine, source can't reach anything else either).

Heuristic 0, *most conservative least generalization from positive instances*, mimics least generalization from closely related, explicitly labeled positive instances. It resulted in very conservative definitions, with relatively little generalization. The positive aspect, however, is that it never produced an over-generalized definition. In general, as more instances are retrieved from memory, the definition becomes more generalized. The number of instances retrieved is limited, though, by the tight restrictions on

Figure 55: Summary of Test Runs for
Network Faults: RULE1

Figure 56: Features to Learn for RULE1

similarity to the trigger instance. Furthermore, in this domain as in others, we experienced what we call the *healthy features effect*. That is, rules in this diagnostic domain primarily look for features with abnormal values. Some combination of abnormal features implies a particular diagnosis. This heuristic will *not* eliminate features that are normal, as they are shared by the instances of a given diagnosis, just as the abnormal features are. Though this may not be incorrect, it is usually undesirable for a domain theory.

In **Heuristic 1**, *least generalization from liberal instances*, the similarity of the subgoals implies that the difference between the explanation retrieved and the trigger instance's lies primarily in the final step to the conclusion. As the difference between the instances is assumed to be isolated to just that one position, the instances are taken to be positive instances of the missing concept. They are, in fact, considered sufficiently similar that they should be covered by a single non-disjunctive definition. This heuristic resulted in a bit more generalization than did the previous one, due to the loosened constraints on similarity for retrieval. That is, the heuristic could find a larger *variety* of instances to which its actions could be applied. It was still, however, affected by the healthy features effect. A similar result was observed for **Heuristic 3**, *conservative least generalization*. As is shown in Figure 55, heuristic 1 generalized the learned rule more quickly (i.e., with fewer instances). Neither of the heuristics was ever observed to over-generalize.

Heuristic 2 and **Heuristic 4**, *different goals mean no relationship regardless of other similarities*, treat the instances retrieved for induction as strongly negative examples. As might be expected, they performed extremely poorly. They eliminated from the new rule *all* features that appeared in the original deleted rule. As can be seen in Figure 55, features 1, 6, and 12, the features in the left-hand-side of the original RULE1, do not appear in the rules learned by these two heuristics.

No instances were retrieved by **Heuristic 5**. That is, heuristic 5 found no instances in memory to which its inductive actions could be applied.

Heuristic 6, *very liberal generalization from positive examples*, states that if instances of the same missing concept look alike on the surface, then they can be covered by a single non-disjunctive definition. Furthermore, features already appearing in their respective explanations can be ignored with respect to the new rule. Though one might expect failure of this heuristic due to the fact that it ignores information about differences from an explanatory analysis, we found that it worked quite well in this case. It generalized more quickly than the more conservative least-generalization heuristics (0, 1, and 3), but it did not over-generalize.

Heuristic 7 was perhaps the most interesting heuristic in this test set of test runs. It generalized extremely quickly, eliminating many of the problematic healthy features. It eliminated seemingly important features, however. On closer examination, we observed that it consistently eliminated features 1 and 6. Though feature 1 (source can't connect) appears in the original rule, it is essentially useless. It simply indicates that a fault exists, and thus gives no additional information when considering many instances, *all* of which are faults. It is also likely that feature 12 (source can't reach anything) generally gives sufficient information without requiring feature 6 (target looks fine) as well. If a user attempting to connect to another net cannot seem to reach *any* net, then it is likely that it itself is the problem, rather than everyone else experiencing problems and the initiator being fine. As we will describe later in Section 5.6, we ran tests to determine the classification accuracy of learned rules on new input instances. Based upon these tests, we found that heuristic 7 did produce good rules. We still take the initial rule to be a good goal for which to strive in general. But this case points to a potential application of a rule learning system such as Gemini. It can be used to refine a rule base. Rules may be deleted and then recreated. Rules that are close to, but not precisely the same as, deleted rules may be more closely analyzed for new insights.

5.4.1.2. Experiments for Missing Rule #3

Figure 57: Explanation using Network Fault Domain RULE3

This section describes test runs where the rule labeled RULE3 was removed from the domain theory for re-learning. Unlike RULE1, discussed in the previous section, RULE3's left-hand-side (lhs) refers to both an input feature and an intermediate concept, *looks-like-source-prob*. This makes the problem of new rule induction slightly more complicated. First, Gemini must consider not only individual features as candidates for inclusion in the lhs of the new rule, but must also consider how combinations of those features might be more concisely described by another concept. Gemini's method for dealing with this was discussed in Chapter 3. A second potential complication arises due to the fact that a greater amount of information might be missing from the partial explanation than simply the missing rule. For example, in our test runs we assume that explanations are generated backwards, working from the goal concept to the input features. Once the explanation generator ''gets stuck'' in attempting to prove an intermediate concept, it can go no farther along that branch of the partial explanation.

Thus, the partial explanation contains less information than it would if the missing rule were the last in the branch. Figure 57 gives the partial version of the explanation shown earlier in Figure 54, this time missing RULE3.

The memory built in order to perform the runs for this rule contained the same 90 instances as for RULE1, reflecting, however a missing RULE3. Eighty of the 90 instances were missing RULE3, just as they were missing RULE1 above. Of these, we selected 10 as trigger instances for induction. As for the earlier deleted rule, the frame designated as the trigger frame was removed from the memory for the duration of the time it acted as trigger.

Figure 58 summarizes the results of applying each of our test heuristics to learning RULE3. Because RULE3 contains both a feature and an intermediate concept, the table is slightly different from that for RULE1. Each table entry gives four numbers. The first two are the number of extra features and the number of missing features in the new rule. The third and fourth give the number of extra intermediate concepts and the number of missing intermediate concepts, respectively. The rightmost column lists the features mentioned in the final rule for the row. It also includes any intermediate concepts that appear in the learned rule. Features are listed by their number codes, as given in Figure 52. Intermediate concepts in the learned rule are referred to by their names. As shown in Figure 59, the feature to be selected for this rule is 22 (the user is on a TRN, that is, a token ring network). The intermediate concept necessary for the original RULE3 is *looks-like-source-prob*.

The behavior of the heuristics in this set of test runs was very similar to that for RULE1, which is not surprising given that essentially the same instances with nearly identical explanations are playing important roles here. The most significant thing to note here is that the behavior was basically *not* affected by the fact that RULE3 would fall in the middle of an explanation rather than at the leaves. One heuristic for which there was an obvious difference was heuristic 9, which eliminates from the new rule all features already in the leaves of the partial explanation. As the partial explanations are shorter here, fewer features can be generalized out of the left-hand-side of the learned

Figure 58: Summary of Test Runs for
Network Faults: RULE3

rule. Another heuristic that behaved differently was heuristic 7, which eventually eliminated everything from the new rule. This probably occurred because of the small

Figure 59: Features to Learn for RULE3

size of the partial explanations in this case. Heuristic 7 is an extremely strong heuristic that eliminates *all* similarities between instances whose explanations look different. Because the explanations were so unusually small here, many other (more complete) explanations looked different. If more information had been available in the partial explanation, those differences might not have appeared.

5.4.1.3. Experiments for Missing Rule #9

This section describes test runs where the rule labeled RULE9 was removed from the domain theory for re-learning. Like RULE3, RULE9's left-hand-side refers to both input features and an intermediate concept, *looks-like-source-prob-TRN*. Because of its relative position in a typical explanation, partial explanations requiring RULE9 contain even less information than those needing RULE3. This is illustrated in Figure 60, which shows the partial explanation missing RULE9, comparable to the explanation of Figure 57.

The memory built in order to perform the runs for this rule contained the same 90 instances as for RULE1 and RULE3, reflecting, however a missing RULE9. A total of 30 of the 90 instances were missing RULE9. Of these, we selected 5 as trigger instances for induction.

Figure 61 summarizes the results of applying each of our test heuristics to learning RULE9. As for the previous rules, the rightmost column indicates input features appearing in the learned rule. Due to the large number of extraneous intermediate concepts in many cases, intermediate concepts in the learned rule are not given here. As

Figure 60: Explanation using Network Fault Domain
RULE9

shown in Figure 62, the instance features to be selected for this rule are 5 (PC initialization failure) and 25 (user says they can't initialize).

A number of observations can be made here. First, the heuristics performing correctly thus far (0, 1, 3, 6, 9), continued to work here, where they applied. They behaved in the same manner, eliminating different sets of extraneous features, but generally having problems with healthy features. In general, fewer heuristics applied, and where they did they did not generalize away as many extraneous features. This is precisely what would be expected in a case where there is less explanatory knowledge available to use.

One interesting heuristic to note in this example is heuristic 5, the *disjuncts are different* heuristic. It found, with very few instances, precisely the two features (5 and 25) to include in the missing rule. The problem it had was in forming a connection to the rest of the explanation (via an intermediate concept). This is not surprising given that it is attempting to distinguish between instances as much as possible, even though at a higher level in the explanation trees they are the same. This could probably be corrected quite easily by performing minimal forward chaining from the original input instance features and using the derived concepts in the new rule. In general we will observe this heuristic performing well where it applies.

Figure 61: Summary of Test Runs for Network Faults: RULE9

Figure 62: Features to Learn for RULE9

5.4.2. Radio Fault Diagnosis: Military / Manpack

Figure 63: Encoding of Manpack Radio
Fault Attributes

This domain involves diagnosis of a communication radio used by the military. It was described in Chapter 4, where we used it to illustrate the application of contextual heuristics for rule learning. As in the network fault domain, encoded knowledge enables isolation of a fault to a particular major radio component, but does not allow deeper analysis of the problem.

Input frames describing fault instances in this domain are single-level frames containing 21 binary-valued features. An input description may be incomplete. The complete set of input frame attributes is given in Figure 63, numbered for later reference.

The complete rule base for this domain contains 35 rules. We deleted two rules to perform test runs characterizing the behavior of the learning heuristics in this domain. The two rules are given in Figure 64. In general, the manpack diagnostic explanations are disjoint in that different diagnostic groups do not share rules. Therefore, we selected rules from different diagnostic groups for testing. The following two sections describe the runs performed for each of these rules. Each of the rules is discussed in more detail in its respective section.

Figure 64: Rules Deleted for Testing in Manpack Radio Fault Diagnosis

5.4.2.1. Experiments for Missing Rule #1

Figure 65: Explanations using Manpack Radio
Fault Domain RULE1

This section describes test runs where the rule labeled RULE1 was removed from the domain theory for re-learning. The left-hand-side of the rule refers only to input attributes, so it would always appear at the bottom of an explanation tree. Sample explanations are given in Figure 65, the first of which is a typical explanation using the rule, and the second of which is the partial explanation derived when the rule is missing.

In performing the runs, Gemini has access to a memory of instances which the system has already analyzed. Heuristics make use of the memory by retrieving specific instances deemed to be useful for induction of the missing rule. The memory of instances used in the test runs here contained 50 instances. Eight were partial as they required RULE1 in order to complete them. Of these, 4 instances were selected as trigger frames for induction.

Figure 66 summarizes the results of applying each of our test heuristics to learning RULE1. As in Figure 55, it gives the number of extraneous features, the number of missing features, and the range of numbers of extraneous features. As shown in Figure 67, the goal of the learning heuristics is to select only feature 1 for the lhs of the new rule.

This domain is clearly much simpler than the previous one in terms of its input descriptions. The frames tend to be small (i.e., contain a small number of features), and thus an inductive system attempting to select features implying a missing concept only has to select from a very limited set of feature choices. Like the previous domain, however, this one involves diagnosis. Selecting features that imply a concept might result in identifying both healthy features and the buggy ones that actually imply the problem. A rule base designer might not want the domain theory to contain information about healthy features. A heuristic such as heuristic 5 (*disjuncts are different*) can help to eliminate problematic healthy features by not only finding a definition that distinguishes between positive and negative instances, but by liberally cutting features to maximize that difference. It is able to do so based upon explanatory information that provides evidence that two instances are unrelated. As can be seen in Figure 66, heuristic 5 goes a bit too far in this case. After retrieving more than 5 instances for induction, it eliminates everything from the new rule. As will be seen in other cases, heuristic 5 generally performs well as long as very few instances are retrieved from memory. This holds true regardless of the specific instances retrieved.

Figure 66: Summary of Test Runs for Manpack Radio Faults: RULE1

Figure 67: Feature to Learn for RULE1

5.4.2.2. Experiments for Missing Rule #24

This section describes test runs where the rule labeled RULE24 was removed from the domain theory for re-learning. As with the previous rule, the lhs of this one refers only to input attributes. Sample explanations are given in Figure 68, the first of which is a typical explanation using the rule, and the second of which is the partial explanation derived when the rule is missing.

The memory of instances used in the test runs here contained the same 50 as for RULE1. Eleven were partial as they required RULE24 to complete them. Of these, 5 instances were selected as trigger frames for induction.

Figure 69 summarizes the results of applying each of our test heuristics to learning RULE24. As shown in Figure 70, the goal of the learning heuristics is to select only feature 6 for the lhs of the new rule.

Results with this example were consistent with earlier cases. Heuristics 0, 1, 3, and 6, (*generalization from positive examples*), and 8 (*disjuncts are different, but conservatively*) behaved as they had for earlier examples. As has been the case already, heuristic 4, which essentially ignores explanatory information, failed badly. Heuristic 5 performed extremely well in this example, essentially immediately eliminating a problematic healthy feature: feature 15, which indicates whether the comsec cable looks fine. An inductive system that did not use our heuristics would probably never be able to do so. We found similar behavior with heuristic 7 in this case.

5.4.3. Radio Fault Diagnosis: Military / Vehicular

This domain involves diagnosis of a communication radio used by the military. Instead of the manpack radio, this domain models six configurations of vehicular radios, all built around a single specific receiver-transmitter. Any of the six can be represented

Figure 68: Explanations using Manpack Radio Fault Domain RULE24

using the input frame developed for this domain. The input representation is a rough

Figure 69: Summary of Test Runs for Manpack Radio Faults: RULE24

Figure 70: Feature to Learn for RULE24

model of the radio rather than a list of symptoms as is used for the manpack radio. The representation does not, however, model all components of these vehicular radios.

Figure 71: Encoding of Vehicular Radio
Fault Attributes

Input frames describing fault instances are hierarchical frames that describe various radio components. Features may be nominal (i.e., symbolic), numeric, binary, or pointers to subframes. A description of an input frame for this domain is given in Figure 71. We have not represented sample slot values in the figure, as we did not want such detail to obscure the important information there. A sample input frame with slot values is given in Appendix B. The representation shown in Figure 71 is flexible. For any given input radio, more or less detail can be represented by describing more or fewer radio parts. The input instances on which we performed our tests, however, were all consistent with the representation shown in the figure. As for the previous domains, we have numbered the input features for later reference. Note that slots which are pointers to subframes are not labeled. When a feature is referenced within a rule, it must be bound to an appropriate (sub)frame first. Thus if a rule references a feature, it will reference the appropriate frame as well. As a result, we focus on the non-pointer features themselves.

We deleted two (of 18) rules to perform test runs in this domain. The two rules are given in Figure 72. The rule base for this domain is interesting in that it models a decision network where many (though not all) of the final diagnoses are single branch points off of a single line of reasoning. Only one rule in the domain theory refers entirely to input features. Once a decision has been made with this rule, reasoning proceeds using rules that relate other features with the current diagnostic hypothesis to determine a next hypothesis, and so on. The first rule we deleted for testing, RULE1, is the single domain rule that refers entirely to input features in its left-hand-side. The second rule we deleted, RULE4, refers to an intermediate conclusion (a possible vehicle battery problem) as well as to an input feature. The following two sections describe the runs performed for each of these rules.

5.4.3.1. Experiments for Missing Rule #1

This section describes test runs where the rule labeled RULE1 was removed from the domain theory for re-learning. Figure 73 gives a sample explanation in which the rule has been used. Each level of inference in the tree is a conjunction (though for clarity we did not explicitly indicate this in the figure).

Figure 72: Rules Deleted for Testing
in Vehicular Radio Fault Diagnosis

The memory of input instances used by Gemini in order to test the heuristics in this domain contained 50 instances. As discussed above, RULE1 is the *only* rule in the domain theory that refers entirely to input features, rather to an intermediate inference. That is, it is the only rule that can appear at the leaves of any explanation tree. Thus after RULE1 was removed from the rule base, every instance in memory had a partial explanation. Of these, 7 were selected as trigger frames for induction, 1 from each major diagnostic group.

Figure 74 summarizes the results of applying each of our test heuristics to learning RULE1. Figure 75 shows that the goal of the learning heuristics is to select features 8 and 9 for the lhs of the new rule. The other condition shown in Figure 75 simply creates a binding to the appropriate nested frame within the input. The features referenced in the rule are those within the bound subframe. Gemini automatically creates bindings to correct subframes.

One difference between these results and those given earlier is immediately

Figure 73: Explanations using Vehicular Radio Domain RULE1

obvious: when triggered by a particular instance to perform induction of a missing rule, only half of the heuristics found additional instances in memory which were related to the trigger instance in the appropriate way for the heuristic action to be applied. This is largely a result of the unusual structure of the domain theory. It cannot generate explanations that are entirely disjoint. This is a positive result for our heuristics, however. Recall that heuristics 5, 7, and 8 use information about dissimilarity of explanations to classify previous instances as negative examples of the concept to be learned. These heuristics found no instances that could be classified as negative. This is

Figure 74: Summary of Test Runs for
Vehicular Radio: RULE1

Figure 75: Features to Learn for RULE1

exactly correct. There were no negative instances in the memory. Every instance in this domain must make use of the missing rule.

Because all instances in this domain are positive examples of the missing concept *poss-ds1-lamp-prob*, the liberal generalization strategy of heuristic 6 worked best.

5.4.3.2. Experiments for Missing Rule #4

Figure 76: Explanations using Vehicular Radio Domain RULE4

Here we summarize the test runs where RULE4 was removed from the domain theory. This rule, unlike the previous one, appears in the middle of an explanation tree as it includes an intermediate concept in its lhs. Figure 76 shows two explanations. The first is a typical explanation using the rule, while the second, labeled (b), shows the partial explanation derived when the rule is missing from the domain theory. Note that there is essentially no partial explanation. As implemented here, explanation generation is backward chaining which stops when an impasse is reached. In this case, the impasse is reached in the first step.

The memory of instances used in the test runs here contained the same 50 as for RULE1. Of these, 6 had partial explanations as they required RULE4. Of the 6, 3 were selected as trigger instances.

Figure 77 summarizes our results for this rule. As shown in Figure 78, the feature to be selected by induction for the new rule definition is 24, which refers to the value of the battery's positive terminal voltage. A new rule must also include a reference to the intermediate concept *poss-vehicle-battery-prob*.

Induction of the new rule (or, more generally, of the definition of a faulty vehicle battery) is more complicated than previous rules for two reasons:

1. it refers to a numeric value rather than to a binary or nominal one;
2. it requires generalization to a range of numbers (i.e., less than 22).

Gemini's rule induction component does perform generalization of feature values. However, it does not handle numeric ranges. It generalizes to a single number. Thus to match the lhs of a learned rule with this type of feature, an instance would have to have a value ‘‘close to’’ the value in the learned rule.

In many ways, this example covers the opposite of the situation for RULE1. Here, when an explanation is partial, it essentially doesn't exist. Recall that explanations are generated by chaining backward from the goal. When an impasse is reached, generation stops. In this case, the impasse is reached immediately and no explanation is produced at all. This is *not* a common situation. The heuristics that classify instances as positive for induction, specifically 0, 1, 3, and 6, require that explanation components match the trigger to varying degrees. Because of the unusual nature of the partial explanations in this case (i.e., that they are empty), previous instance explanations will either match the trigger instance explanation exactly (they will also be empty), or they will not match at

Figure 77: Summary of Test Runs for
Vehicular Radio: RULE4

all. Thus the only heuristic classifying instances as positive for induction that applied here was heuristic 0.

As in the network fault domain, heuristic 0 performed reasonably well, generalizing in the direction of the required definition without over-generalizing. It resulted in rules that referenced the appropriate intermediate concept in their left-hand-sides. And it also included the required feature, specifying that its value be one in the original range of values. It included extraneous features, which would make new rule

Figure 78: Features to Learn for RULE4

application more specialized. All of the extraneous features, however, were ‘‘healthy features’’. Recall from earlier discussions that many diagnostic rules look for features with abnormal values. Some combination of abnormal values implies a particular diagnosis. Sometimes learned rules also look at healthy features that are shared by the instances of a given diagnosis. Though not incorrect, this is not always desirable.

Heuristic 5, the *disjuncts are different* heuristic, gave interesting results here. Though the learned rules appear to be wrong in that they do not include the reference to the appropriate feature, they include an alternative feature that also indicates a problem with a vehicle battery. That is, the learned rules for a vehicle battery problem refer to an incorrect voltage for the negative terminal, whereas the removed rule referred to an incorrect voltage for the positive terminal. Incorrect voltages on either of the two are equally problematic. Heuristic 7, which also finds negative instances for induction, behaved similarly. In fact, both converged to good definitions with very few examples. They generalized the new rule substantially with only a single instance retrieved from memory for induction. A problem, however, was that they eliminated the reference to the missing intermediate concept, *poss-vehicle-battery-prob*. A solution to this problem was introduced briefly in the discussion of network fault RULE9. The system could perform minimal forward chaining from the original input instance features, and could use the derived concepts in the new rule.

5.4.4. Terrorist Event News Stories

Figure 79: Encoding of Terrorist Event Story Attributes

The final domain to be discussed in this chapter is the terrorist event news story domain. This domain, as introduced earlier, is modeled loosely around the following performance task: given information from a news story describing a terrorist event, suggest an action that a law enforcement agency might take in response, given a database of earlier events. Input frames in this domain are translated by hand from newspaper stories into Gemini's representation. Gemini's goal is to explain why each event is a terrorist event, as opposed to some other type of news event. Though this is a classification domain, it differs from the other three in that it does not involve diagnosis. It also differs from the others in that the input instances are much more varied. Input instances in this domain are hierarchical frames, like the vehicular radio domain. Unlike that domain, however, input instances cover a wide range of the frame configurations allowed by the representation. We show one example, repeated from Chapter 3, in Figure 79. Because there is such variability in the possible inputs, a set of allowable instance features cannot be easily labeled as in the previous domains. The input of Figure 79, called event23, will be used to describe the behavior of the heuristics in this domain. References to input features by codes will refer specifically to event23's features.

The rule base for this domain contains 53 rules. We deleted two rules to perform test runs characterizing the behavior of learning heuristics in this domain. The two rules are shown in Figure 80. The following sections describe results of test runs in which each of the two rules was deleted individually, and then results where the two rules were deleted at once.

Figure 80: Rules Deleted for Testing in Terrorist Event News Story Analysis

5.4.4.1. Experiments for Missing Rule #50

This section describes test runs where the rule labeled RULE50 was removed from the domain theory for re-learning. The left-hand-side of the rule refers to both input features and an intermediate concept. Sample explanations using the rule are given in Figure 81. The explanation marked (a) is the complete explanation for event23; (b) shows the partial explanation derived when RULE50 is missing from the domain theory.

In performing the runs, Gemini had access to a memory of 50 instances. Of these, 8 required RULE50 to complete their explanations. Four were selected as trigger instances. Figure 82 summarizes our results for this rule. As shown in Figure 83, the goal of the learning heuristics is to create a left-hand-side with features 5 and 6, and with intermediate concept *not_significant_warning*.

Clearly far fewer heuristics could be applied in this domain than could be in the previous three. This is a result of the fact that *all* explanation goals in this domain are *terrorist_event*. That is, all instances are ultimately examples of the same concept. Heuristic 0, which does least generalization from closely related examples, did apply, however. It performed comparably to the other domains. With more examples, it produced more general rules, but it never over-generalized. That is, it never eliminated a feature of the original rule from the new definition. Heuristic 9 also worked correctly, never eliminating necessary features, though it left many extraneous features in the new rule as the explanations were not very large.

Heuristic 5, *disjuncts are different*, performed poorly in this case, as opposed to the way it performed in previous cases.

Figure 81: Explanations using Terrorist Event RULE50

Figure 82: Summary of Test Runs for Terrorist Events: RULE50

5.4.4.2. Experiments for Missing Rule #41

Here we describe test runs where RULE41 was removed from the terrorist event domain for re-learning. Unlike the previous rule, this one refers only to an input feature and therefore falls at the leaves of an explanation tree. This was shown in Figure 81. Figure 84 shows the corresponding partial explanation derived for event23 when RULE41 is missing.

Figure 83: Features to Learn for RULE50

Figure 84: Partial Explanation Missing
Terrorist Event RULE41

The memory of instances used in the test runs here contained the same 50 instances as for RULE50. Seven required RULE41 to complete them, and 4 of these were selected as trigger instances for testing. Figure 85 summarizes our results for this rule.

As shown in Figure 86, the only instance feature to be selected for inclusion in the new rule was feature 1, indicating an explosion.

Figure 85: Summary of Test Runs for Terrorist Events: RULE41

Figure 86: Feature to Learn for RULE41

Results for this rule were consistent with the results for RULE50. This is also an interesting example for its role in identifying an additional use for Gemini. As indicated earlier, we created the terrorist event rule base as an early test set for Gemini. One rule that we originally encoded inferred an explosion from facts about bombs not being defused and no warning being given, among other requirements. When that rule was removed from the domain theory in order to test the rule induction component, Gemini created a rule that inferred an explosion from the feature stating that there had been an explosion! We had missed the fact that there was no need to make an inference about an explosion. The information was already available and could simply be extracted from the input frame. Gemini, therefore, aided us in knowledge engineering. It was that experience that led to the evolution of RULE41.

We performed one final set of test runs in the terrorist event domain in which we focused on learning RULE41, but in the context of RULE50 missing as well. Our hypothesis was that there would be very little difference, except that the lack of RULE50 would result in a bit less explanatory information for the heuristics. This hypothesis was correct, as shown in Figure 87. The chart gives the results for RULE41 with RULE50 missing as well. The results are clearly almost identical to those in Figure 85. This will not necessarily always be the case. We expect to find this behavior when there are few rules missing in an explanation and when they are missing relatively close to the leaves. When many rules are missing, or when they are missing very high in the explanation tree, the lack of explanatory information can result in bad retrievals by the heuristics.

We noted earlier that fewer heuristics could be applied in this domain than could be in the other three. This occurred because *all* explanation goals in this domain are *terrorist_event*. That is, all instances are ultimately examples of the same concept. Any heuristic that looks for instances of different goal concepts would therefore not apply. Our belief is that this would generally not be a problem. Most rule bases for classification contain a number of final class possibilities. In fact, we could have effectively encoded our domain theory to classify instances as one of a number of different terrorist event types, rather than explaining that they are terrorist events in general.

Figure 87: Summary of Test Runs for Terrorist Events: Both Rules Missing

5.5. A Discussion of Heuristic Performance

The preceding sections discussed the performance of our inductive heuristics in four individual domains. This section summarizes the results from the perspective of the heuristics.

Heuristic 0 performs least generalization from a set of instances determined to be very closely related positive instances. This heuristic applied in all domains and behaved in an expected manner. Since all instances were required to be positive examples of the missing concept and since they were all very nearly identical to the trigger instance with respect to explanatory structure, they were generally coverable by a single conjunctive concept definition. This is essentially a non-heuristic, performing the most conservative generalization possible. One problem with this heuristic, other than its slowness in generalizing, was that it suffered from the healthy features effect described in Section 5.4.1.1.

Heuristic 1, *least generalization from liberal instances*, performs generalization from positive examples, but classifies instances as positive based upon more relaxed criteria than heuristic 0. Though the heuristic specifies that retrieved instances must be examples of the same missing concept, it has no guarantee that a single conjunctive definition will be sufficient for coverage. It is possible that the concept has a disjunctive definition, and that the instances should be classified by different disjuncts. Because the conditions on applying the heuristic also require that the instances share their proven subgoals, it assumes that the single definition will be sufficient. In the one domain where this heuristic applied (network fault diagnosis), we found this to be the case. That is, the conditions on applying the heuristic were sufficiently strong to retrieve instances that were all members of a single disjunct. We would *not* expect this to be true in general. This heuristic also suffered from the healthy features effect.

Heuristic 2 considered the same instances as the preceding one, but treated them as strongly negative examples of the missing concept. This heuristic was examined to test our intuition of what would be a bad rule of thumb. As expected, it performed poorly in the network fault domain, the single domain in which it applied. In that domain it eliminated virtually everything important from the new rule and retained many extraneous features.

Heuristic 3, *conservative least generalization*, had wide applicability, being useful in almost all cases. This heuristic retrieves positive instances of the missing concept, but allows them to be members of different final goal concepts. One might expect such instances to be covered by a disjunctive missing concept definition. We found, in our domains, that a single non-disjunctive definition was sufficient in all these cases. This heuristic, like heuristic 1, never over-generalized, but generalized more quickly than heuristic 0.

Like heuristic 2, **Heuristic 4** was tested as an example of a rule that we expected to fail. Even more so than heuristic 2, it tended to eliminate all important features immediately from the definition being learned.

Heuristic 5 exhibited extremely interesting behavior. This is a version of the ‘‘disjuncts are different’’ rule discussed at various points throughout the text. The idea behind ‘‘disjuncts are different’’ rules is the following: if a goal concept is explained in different (disjunctive) ways, then the difference should be maintained in any missing parts of the explanations as well. While working well in the two radio domains, it eliminated *all* features from the definitions for other rules in the network domain. We observed, however, that this did not occur immediately (i.e., when only small numbers of instances were retrieved for induction). In fact, the heuristic generalized extremely well with small numbers of instances, often requiring just one to eliminate the healthy features problem.

Heuristic 6, *very liberal generalization from positive instances*, was not applicable in all cases, but where it was it generally retrieved large numbers of instances. This heuristic treats all positive instances of a missing concept as closely related instances provided they look similar in their input representations. We observed this heuristic to work quite well, though we would not expect it to do so in all domains. Many domains have greater numbers of disjunctive rules and this heuristic concentrates on creating a single conjunctive definition for all instances of the missing subgoal. Outside of the network domain, however, it suffers from the healthy features problem.

Heuristic 7 was among the most generally applicable of the heuristics. Another variant of the ‘‘disjuncts are different’’ rule, it exhibited similar behavior to heuristic 5. The primary difference between the two was that heuristic 7 applied more often, and when it did, it was able to retrieve more instances for induction. Though it performed fairly well, it ignores much information available from instances themselves. It focuses exclusively on explanations to determine the relationship between instances. This could be problematic.

Heuristic 8 exhibited behavior similar to 0, 1, 3, and 6. It never over-generalized. Improvement was observed as more instances were retrieved, but it appeared to happen quite slowly.

Heuristic 9 removed all features from a rule that were present in the explanation already. Though this is a seemingly dangerous heuristic to apply, we found it to work correctly in all of our domains. The amount of generalization obviously depends upon the size of the partial explanation. Even when small, however, this is a worthwhile heuristic to apply as it requires essentially no resources.

In general, a combination of using one of {0, 1, 3, 6, 8}, one of {5, 7}, and heuristic 9 are an effective combination for induction:

- Heuristics 0, 1, and 3 are effective in labeling instances as positive for induction. They never resulted in over-generalization, though they resulted in some over-specialization. Heuristic 6 also worked well, though it was better at generalizing. Its conditions for application might be a bit too liberal, however, in that they classify essentially all instances of a concept with a single non-disjunctive definition. Heuristic 8 was successful as well.
- Heuristics 5 and 7 treat instances as strongly negative. They perform extremely well in generalization, but when *only very few* instances are retrieved. Heuristic 5 is probably safer than heuristic 7.
- Heuristic 9 assumes that features will appear at most once per explanation. It worked appropriately in all of our domains.

A combination of heuristics is more effective than any single heuristic as different heuristics focus on different sets of instance features. The heuristics can be applied in any order. That is, they are not subject to order effects. When a heuristic searches through memory for earlier instances, it *always* does so with respect to the trigger instance, regardless of any progress that may already have been made in generalizing a new rule.

5.6. Evaluation of Learned Rules (Revisited)

Throughout this chapter we have discussed the quality of newly learned rules with respect to the rules deleted from the domains theories. This is a reasonable criterion since three of the four domains in which we have tested our heuristics were encoded by experts. Other potential criteria for judging new rules exist, however. One of these involves testing the performance of learned rules. That is, for a given original rule, a number of test instances are classified as positive or negative instances with that rule. Then the test instances are classified again with the learned rule. The classifications with the learned rule are then compared to the (presumably correct) classifications of the original rule.

We evaluated one set of learned rules on the basis of their classification performance. We tested all the rules learned when RULE1 was deleted from the network fault domain. RULE1 was shown in Figure 56. Figure 88 summarizes our test results in a chart. Each row in the chart gives the performance results for the rules learned using one heuristic. Each heuristic was evaluated using 10 different trigger instances as in the earlier test runs. For each heuristic we give the results using one trigger instance displaying median behavior. In the chart, each entry gives the number of instances that were correctly classified using the rule learned by a particular heuristic with a specific number of instances. There were 50 instances classified for each rule. The test instances were different from the instances used in training.

We make a number of important observations in comparing Figure 88 and Figure 55. Figure 55 summarized the performance of the heuristics in terms of the features appearing in the learned rule. Each entry in that table lists the features in the learned rule, while the corresponding entry in Figure 88 gives the classification accuracy of that rule. We observe that:

- The results shown in the two charts are analogous. Our criterion for evaluating the results of Figure 55 was that the learned rule should refer to the same features as the missing rule. Thus those learned rules with fewer extraneous features are better than rules with more extraneous features. We see that as the number of extraneous features decreases, the classification accuracy increases.
- The only cases in which there is some discrepancy are heuristics 2 and 4, which we had found to behave poorly in the earlier analysis as they eliminated important features from the learned rules.

Thus, in general, our earlier evaluation criterion, which focused on matching the features referenced in a missing rule, reasonably mirrors the classification accuracy measure. In addition, we observed that our learned rules *never* resulted in any false positive classifications. That is, no negative instance was ever classified as positive. This means that, in general, the learned rules were too specialized. If rules are too specialized, future instances might not be explainable. In those cases, new (more general) rules can be learned. No rule will have to be retracted, however, as one that is incorrect.

Figure 88: Evaluation of New Rule Performance

Though additional performance analysis would be interesting, the advantages of our first evaluation criterion are that:

1. It reflects the performance of the learned rules with respect to classification accuracy.
2. It requires that learned rules not only work well in terms of performance, but that they be conceptually easy to read. This is ensured by the fact that they are required to mimic the rules of experts.

5.7. Summary

This chapter has presented tests performed to evaluate a set of contextual heuristics for rule induction. Contextual heuristics are important for inductive learning of rules missing from a domain theory. They allow a system to exploit existing knowledge in order to select its own instances for induction. They also help to generalize learned rules. Thus the system is capable of filling gaps in its knowledge base without relying on a human for additional guidance or for auxiliary knowledge.

The heuristics we tested were selected from those generated using the methodology of Chapter 4. They were chosen as a representative set of heuristics in that some were hypothesized to work well and others were hypothesized to work poorly. Each heuristic was tested in four domains.

We have observed that many heuristics are effective in guiding induction of correct rules. Heuristics 0, 1, 3, and 6, which classify previously seen instances as positive for induction are effective, as are heuristics 5 and 7, which classify instances as negative for induction. Heuristic 9, which eliminates previously used features, was shown to perform correctly as well. In addition, heuristic 9 requires virtually no resources in order to be carried out.

Because heuristics essentially classify instances as positive or negative for induction of a specific concept (the missing subgoal), no outside intervention is required to label the instances. In addition, a heuristic indicates the strength of its classification by indicating whether attention should be focused on all features or on only the explanation leaves. Heuristic 8, for example, is conservative in that it eliminates only leaves from the new rule.

Heuristics behave independently and focus, in general, on different subsets of instance features. Chapter 4 traced through an example to which a number of heuristics were applied in sequence. A good rule to follow for the application of heuristics is to use one of {0, 1, 3, 6, 8}; then use one of {5, 7} to eliminate ‘‘healthy features’’; then use 9 to eliminate any additional extraneous features.

Chapter 6

An Analytical Evaluation of the Research

6.1. Introduction

This chapter outlines the major contributions of our research, and analyzes it with respect to other closely related work. We have devised a solution to the problem of EBL domain theory incompleteness which involves the extraction of explanatory knowledge to guide induction of missing rules. Given an input example for which only a partial explanation can be derived, we define the problem of incompleteness to be induction of a missing concept definition. We have shown that knowledge can be extracted from the partial explanation as well as previously seen explanations to guide the induction of the missing rule. Explanatory knowledge provides evidence for earlier instances to be treated roughly as positive or negative examples for induction. In addition, it can be used to further generalize a newly induced rule when abundant examples are not available. This is a significant departure from earlier work in that:

1. Our solution makes use of available information rather than expecting that additional information will be provided by an external source. This results in greater autonomy.
2. The method makes use of heuristic information, but does not require any additional knowledge that must be assumed complete and correct for it to work.

Chapter 5 provided evidence that our method can be successfully applied. This chapter provides a comparison with related work. Section 6.2 discusses the contributions of the research in greater detail, summarizing the major differences between our work and earlier research in this area. Section 6.3 describes the methodology we follow in evaluating our work with respect to that of others. Section 6.4 presents the evaluation. We conclude in Section 6.5 with a summary.

6.2. Contributions of this Research

Our research makes contributions to two distinct areas of machine learning: explanation-based and similarity-based learning. The following is a summary of those contributions:

- **Explanation-based learning:** We have devised and implemented a solution to the problem of domain theory incompleteness that relaxes significant restrictions of earlier solutions.
  1. We do not require an auxiliary domain theory.
  2. We do not require interaction with an expert.
  3. We focus on learning specific missing concept definitions to ensure the conceptual integrity of the augmented domain theory.
- **Similarity-based learning:** When used to address concept learning in the context of an existing partial theory, we have shown that SBL can make use of information already available in the partial theory.
  1. We have shown that induction of missing rules can be accomplished by considering examples already available to an EBL system, even if they are not labeled as positive or negative for induction of a specific concept.
  2. We have shown that general information available in EBL can be extracted and applied automatically to select examples for induction. As a result, an EBL system missing rules need not rely on an external source (human or otherwise) to provide more information.
  3. We have shown that explanatory information can be used to generalize rules even when examples are not available.

  We have shown that contextual heuristics that make use of explanatory information can be automatically generated by considering general attributes of EBL systems. They need not be devised by a system implementer.

  We have identified a number of useful heuristics, and have devised a methodology for evaluating heuristics using Gemini. The ability to explicitly evaluate knowledge used by an inductive method is important.
- **General:** In implementing Gemini, we have shown that inductive and deductive machine learning techniques can be successfully integrated for the mutual benefit of the methods.

Ours is not the first solution to the problem of domain theory incompleteness or, more generally, to the problem of induction. The principal improvement of our work over others' is that we have shown that the problem can be solved by using information already available to a system rather than by relying on an external (human) source either for additional examples or for additional theories.

Consider inductive concept learning. In learning-from-examples induction, examples labeled as positive and/or negative are input to a system. We have shown that examples input to a system can be effectively stored for later retrieval, when the induction of a new rule rule is required. Gemini makes use of explanatory information to label the retrieved examples itself, rather than relying on an external (generally human) source to do so. This means that it is able to learn without additional input from the environment.

Gemini also uses explanatory information to guide generalization of learned rules without requiring additional examples. Recall the contextual heuristic that generalizes a new rule so that its conditions need not mention features already referenced in a partial explanation. This heuristic, and others like it, lessen the number of examples required for induction by focusing directly on information provided by an explanation.

Since applying standard inductive techniques to the problem of domain theory incompleteness suffers from the need for explicitly labeled examples, some researchers have devised methods that make use of auxiliary information. In some of these, auxiliary information takes the form of an additional domain theory. In these solutions the additional domain theory is assumed to be complete and correct, and thus the approaches suffer from the same unrealistic assumption that they are attempting to overcome. Our approach makes use of an additional knowledge source to select examples for induction: a set of contextual heuristics. However, the nature of heuristics is that they provide a useful guide, but need not be assumed to be complete. We have empirically shown contextual heuristics to be effective. Furthermore, though different sets of heuristics will be more or less effective in particular domains, the heuristics are domain-independent.

As a primary goal of this work is to minimize the burdens placed on a human expert, we give algorithms for generating contextual heuristics and evaluating their performance.

In summary, we have described a solution to the incomplete theory problem that makes use of available information in order to minimize the need for additional and/or externally provided knowledge. We have made the realistic assumptions that:

1. A system may need to induce missing rules ‘‘on the fly’’, without being able to consult the environment for more information.
2. A system cannot assume that any knowledge it has is perfect.

6.3. Evaluating the Contributions

Our solution to the problem of domain theory incompleteness, implemented in Gemini, provides a mechanism by which missing knowledge can be determined using information readily available to EBL systems. The solution promotes system autonomy by minimizing its dependence upon a tutor or auxiliary domain theory. In general, Gemini handles the incomplete domain theory problem under a looser set of assumptions than other proposed solutions.

Our comparison to other methods is qualitative. As our technique solves the problem of domain theory incompleteness under a relaxed set of assumptions, there is no equal system against which it can be compared empirically. We enumerate a set of qualitative points for comparison and then discuss how the methods are rated according to those points. For example, we give comparisons of system output representations and domains handled. Section 6.3.1 lists the methods to be evaluated, while Section 6.3.2 discusses the points for comparison. The evaluation is presented in Section 6.4.

6.3.1. Methods to be Compared

Our analytical evaluation of Gemini is intended to show that it solves the problem of domain theory incompleteness under a relaxed (and more realistic) set of assumptions. Though Gemini operates without the restrictions of other systems, its performance remains within acceptable time and space bounds. The methods against which we compare Gemini are those described in Chapter 2:

1. Rajamoney’s ADEPT system;
2. Pazzani’s OCCAM;
3. Hall’s method of Learning by Failing to Explain;
4. Kodratoff and Tecuci’s DISCIPLE;
5. Wilkins’ ODYSSEUS.

6.3.2. Points for Comparison

There are a number of different dimensions along which the methods can be analyzed. We have selected the following as central:

Domain of application - Though each of the methods can theoretically apply to a variety of domains, each is generally best suited to a particular class of domains. We describe the domains to which each technique has been applied.

What is learned? - Though each of the systems compared in this chapter learns knowledge missing from a domain theory, that knowledge can be represented in a variety of ways. This comparison point gives both the knowledge representation learned as well as a conceptual description of the knowledge represented.

Expectations about existing knowledge - In learning missing knowledge, assumptions must be made about the information used in the process of knowledge acquisition. This is true regardless of the source of the information (i.e., whether it is extracted from examples, or whether it comes from an auxiliary knowledge base).

Interaction with a tutor or human expert - A claim of our work is that autonomy is one important goal for machine learning systems to attain. This measures the degree to which a teacher is required to interact with the learning system so that it can properly acquire the rules it needs.

Auxiliary knowledge required - As discussed earlier, a number of solutions to the problem of incompleteness make use of an auxiliary theory. In essence, they are theories about how the missing knowledge can be constructed. In evaluating systems based on this point, we discuss the kind of theory employed, restrictions about the domains to which the theory can be applied, and assumptions about the quality of the theory.

Additional storage required - This is a measure of the amount of space required to store auxiliary information necessary for the method to learn missing rules.

Hypotheses considered - At any time a number of new rules (or hypotheses) may be considered valid given the information available for learning. This is a measure of the number of rules actively considered to be plausible at the time of learning.

Examples required - In completing the explanation for a given input example, a system will often consult other examples. This is a measure of the number of examples from which information is extracted for induction. That is, it gives the number of examples from which a new rule is extrapolated.

Examples considered - Though in some systems this is identical to the metric just described (examples required), this refers more generally to the number of examples that must be looked at for each rule induction. A system might need to consider an example to determine that it provides no useful information for induction. Thus the system spends resources on the example, but the example does not contribute to the actual creation of the new rule.

Time required - This is the order of magnitude of time required to create a missing rule. It includes additional time required for setting up the learning process.

6.4. Analytical Evaluation and Comparison with Related Research

This section compares Gemini to other closely related solutions to the problem of domain theory incompleteness. We compare the systems based upon the points described in the previous section. The chart shown in Figure 89 summarizes the discussion.

6.4.1. Domain of Application

The techniques compared in this chapter have exhibited fairly broad domain coverage. Any method, however, may require making certain assumptions that restrict its applicability. This section summarizes the domains in which each of the comparison methods has been tested, and describes the more general class of domains for which it could be used.

Rajamoney's ADEPT system focuses on experimentation as a method for proposing and verifying rules missing from a domain theory. It has been tested in a

Figure 89: A Comparison of Solutions to the Incomplete Theory Problem

domain describing physical processes, such as evaporation. In general, the method would be expected to apply to any domain in which experiments could be designed and carried out. For some domains this would require a simulation model that might be difficult if not impossible to construct, as for medical experimentation — or, more pertinently, to our diagnostic domains. The technique would not apply to certain classification domains such as the terrorist event news story domain used in our research.

Pazzani's OCCAM makes use of a set of rule templates that describe plausible causal behaviors. Thus his system is restricted to causal domains. OCCAM would not, for instance, be able to find missing rules in our diagnostic domains. In our diagnostic domains, many of the rules relate a set of symptoms to a diagnosis, but they do not necessarily specify a causal relationship. Consider the network fault domain. One rule from that domain states that if a user is failing to establish a connection with a particular node, but the node is responding in general, then the problem probably lies with the initiator of the failed communication. Here the failure to establish a connection is a manifestation of a problem. It is not a *cause*.

OCCAM has been tested on economic sanctions and kidnapping domains, among others.

Hall's method, Learning by Failing to Explain, was applied to the domain of circuit design. It makes use of the idea that two different implementations may have the same function. Using this, analogies are drawn between input examples from which new rules are learned. To apply to other domains, a different definition of analogy would need to be devised.

Kodratoff and Tecuci applied their system, DISCIPLE, to the domain of loudspeaker manufacturing. In general, their method relies heavily on interaction with a human expert, and is therefore somewhat less limited in the domains it can handle. The human expert essentially provides DISCIPLE with the missing rule.

Wilkins has applied ODYSSEUS to the domain of medical diagnosis. Though it assumes interaction with an expert, ODYSSEUS is more restricted than DISCIPLE in that it has encoded in it for analysis not only a domain theory for a particular class of medical problems, but a theory of strategies for making diagnoses. Thus it would require additional modification before it could be applied to other domains. Many of the diagnostic strategies would probably be useful outside of medical domains, however, and would therefore apply to our diagnostic domains.

Gemini has been applied to four domains: network fault diagnosis, manpack radio fault diagnosis, vehicular radio fault diagnosis, and terrorist event news story analysis. As all of these are examples of classification domains, Gemini would be expected to apply to other classification domains as well, such as the diagnostic domain of ODYSSEUS. Gemini could be assumed to have some difficulty, however, in domains requiring procedural or time analysis. It would therefore be less successful with Pazzani's, Rajamoney's, or Hall's domains. In general, those systems work in procedural or time-dependent domains, whereas Gemini and ODYSSEUS work in classification domains.

6.4.2. What is Learned?

Each of the systems compared in this chapter learns knowledge missing from a domain theory. That knowledge, however, can be represented in a variety of ways. In ADEPT, for instance, learned knowledge is some physical process represented using Forbus' Qualitative Process Theory. In OCCAM, causal rules are learned. Hall's system learns derivation rules, where the left-hand-sides are functions and the right-hand-sides are their circuit implementations.

DISCIPLE learns rules for an expert system performing design for loudspeaker manufacturing. ODYSSEUS learns rules in the appropriate format for the HERACLES expert system shell (essentially the NEOMYCIN expert system with the medical domain knowledge removed). Gemini learns classification rules. Though Gemini does not directly learn rules in a specific expert system representation, it *is* able to learn diagnostic expert system rules. For instance, the network domain in which Gemini was tested was directly translated from an expert system.

6.4.3. Expectations about Existing Knowledge

Every system discussed here clearly relaxes a significant assumption generally made by EBL. That is, that the domain theory must be complete. However, each system makes assumptions about other characteristics of the knowledge. Here we refer to knowledge already existing as part of the EBL system, as opposed to auxiliary knowledge required for the purpose of learning missing rules.

All the methods discussed assume that the domain theory is tractable, and furthermore that there is no noise in the input. That is, when asked to derive an explanation that an example is an instance of a concept, each system assumes that the example is *not* falsely classified. Hall also makes the assumption that the given domain theory is very nearly complete, resulting in the need to learn at most one rule for each partial explanation. Pazzani, Wilkins, and Rajamoney similarly make single fault assumptions. None of the systems except Hall's explicitly requires that the domain theory be correct.

6.4.4. Interaction with a Tutor or Human Expert

A driving force in our research has been the desire to minimize the need for extensive interaction with a human expert. This is also a concern for Rajamoney and Pazzani. Rajamoney's system, ADEPT, generates its own experiments to test hypotheses. Pazzani, in his system OCCAM, generalizes learned rules by waiting for additional examples to enter the system during the normal course of operation.

Hall, Kodratoff and Tecuci, and Wilkins, on the other hand, have devised systems with which an expert can interact to tune a domain theory. Their systems are well-suited for this purpose, but cannot function on their own.

One respect in which Gemini currently requires human intervention is in the selection of ‘‘best’’ partial explanations. Although this is a hard problem, there are already paths of research showing sufficiently promising results [O'Rorke 90] and thus we assume that it will become less relevant a concern in the near future.

6.4.5. Auxiliary Knowledge Required

With the exception of the systems receiving new rules or specially tailored lessons from an expert (Hall's system, DISCIPLE, and ODYSSEUS), all the systems compared here make use of some sort of auxiliary knowledge. ADEPT requires a theory of experiment design, while OCCAM requires a theory of causality. Gemini requires a knowledge base of contextual heuristics. Though the heuristics are an additional source of knowledge, they are significantly different from an auxiliary domain theory. They are domain independent (though there is some variation in their efficacy in different domains). Because they are based on general types of knowledge available in an EBL domain theory, they can be systematically generated rather than painstakingly engineered.

6.4.6. Additional Storage Required

In general, systems requiring auxiliary knowledge incur a cost in that they must store their extra knowledge. Such systems include ADEPT, OCCAM, and Gemini. In addition, Gemini stores all examples ever seen by the system in a generalization-based memory (GBM). These examples are used by the rule-learning component when additional examples are not forthcoming from the environment. The GBM is tree-like in the number of nodes within it (though there may be additional or missing connections among the nodes). The upper bound on its size is $O(n^2)$, where n is the number of input examples. With the examples we store the explanatory knowledge referenced by the contextual heuristics: explanation goals, subgoals, missing subgoals, leaves, and the names of rules used. Complete explanations are not retained.

6.4.7. Hypotheses Considered

A rule can be learned through either data-driven or hypothesis-driven methods. A hypothesis can be proposed and then verified by looking at data, or data can be considered that points to a hypothesis. Rajamoney's approach in ADEPT is to consider a large number of hypotheses, many of which are later ruled out by looking at examples. Gemini and OCCAM, on the other hand, propose a very specific hypothesis on the basis of one example and then generalize it using additional instances. There is no clear advantage to using one approach over the other. The hypothesis-driven approach requires the consideration of many possibilities in parallel. Each possibility may require as much time to evaluate as generating a single hypothesis by other methods. Thus it can be computationally expensive. Furthermore, it can be expensive with respect to the space required to store each possibility. Incremental data-driven approaches, on the other hand, are subject to the problems of hill-climbing approaches. That is, the approach may focus on a local ‘‘best’’ answer, rather than finding the most appropriate one to fit all the data. Pazzani’s OCCAM proposes a new rule based upon the input that required completion. It then retains the new rule for later modification using additional examples. We follow a similar approach in Gemini, but we retain all the examples so that a later rule (which would replace an earlier one) will have a more *global* picture of the data it is to cover.

Hall’s system, as well as DISCIPLE and ODYSSEUS, more quickly focus on a hypothesis since it is essentially given to them by the expert.

6.4.8. Examples Required

In completing the explanation for a given input example, a system will often consult other examples. The number of examples required for learning will obviously vary depending upon the information to be learned and the domain tested, and can therefore not be neatly summarized by a single number. We mention it here to point out that the systems discussed generally require fewer examples than pure inductive learning would, as additional information — whether a tutor or an extra theory — takes the place of some examples. The significance of Gemini is that it makes use of existing contextual information to lessen the number of examples required, rather than relying on an external source (a tutor or extra theory) to do so for it.

6.4.9. Examples Considered

Currently, in order to determine whether a previously seen example matches the condition of a given heuristic, Gemini searches *all* examples in memory. Thus, if n examples have been seen, n are looked at again. This could be implemented differently, but is at least bounded by this number. In this metric, Gemini differs considerably from many of the other systems for rule induction. The most notable difference is that in the interactive systems such as DISCIPLE, ODYSSEUS, and Hall's system, the examples are tailored by an expert and are presumably all valuable for induction of the missing rule.

6.4.10. Time Required

Here we focus on Gemini, showing that its performance is within reasonable polynomial time bounds. This is a worst case analysis of Gemini's behavior when an example requires one missing rule. Recall that in Gemini, when multiple rules are missing, each must be learned separately.

Before any rule is learned, Gemini must first place the example in memory. Say that n examples have already been input to Gemini. In order to incorporate the new example, it must effectively be compared against each instance node in the memory. As the size of the GBM is at most n^2, this step is $O(n^2)$.

Once the example is stored in the GBM, examples are extracted to perform least generalization of closely related instances of the missing concept. This is determined by finding the siblings and close ''cousins'' of the current example in memory. As Gemini maintains bi-directional links between parent and child nodes in the memory, this is a relatively efficient process, $O(n)$ at most.

Now say a heuristic is to be employed to generalize the new rule. For each heuristic employed (say there are y), input instances in the memory must be compared against the heuristic's condition to determine whether it applies. Thus n examples must be considered. For each example fitting the conditions, the example must be considered again in generalizing the new rule.

Each ''consideration'' of an example requires a simple matching operation of its features. Say there are x features. Then the time required to learn a rule is bounded by:

$$xn^2 + n + y(xn + xn).$$

In general, Gemini must perform more work than other solutions to the problem of incompleteness as it must store, classify, and retrieve the examples from which it creates new rules. It must do so to avoid restrictive assumptions made by other systems about the knowledge sources available for learning. In spite of the additional work that Gemini must do, its performance remains within (reasonable) polynomial bounds, even in a worst case analysis.

6.5. Summary

The problem of incompleteness in domain theories is an important one, both in EBL and in rule-based systems in general. Given the desire to automate the process of new rule creation as much as possible, we have devised an approach that applies contextual heuristics to use knowledge already available to the system needing a new rule.

Gemini's approach is clearly different from earlier methods. An important distinction is that it is able to perform induction without constant interaction with a tutor. On the other hand, it does not make the strong assumptions about the quality of its auxiliary theory that systems such as Pazzani's and Rajamoney's do. Furthermore, Gemini can be applied to classification domains that Pazzani's and Rajamoney's systems do not cover.

As Gemini does not make a number of restrictive assumptions required for other methods, one would expect that it's performance would be significantly hindered. We have discussed a number of metrics showing that Gemini performs at least reasonably, in spite of its relaxation of assumptions.

Chapter 7

Conclusions and Further Work

7.1. Summary of the Dissertation

This dissertation presents a solution to the problem of incomplete domain theories. Our solution is successful in avoiding restrictive assumptions made by previous methods. It relies primarily on induction as a learning mechanism, but makes use of knowledge readily available in explanation-based learning to guide induction. We have generated and tested contextual heuristics that make use of explanatory information to label examples as positive or negative for induction, and that otherwise guide the generalization of learned rules.

Our search for a solution to the incomplete domain theory problem was motivated by the desire to make the learning system as autonomous as possible. That is, we wanted to minimize the need for interaction with the environment (in most cases, a human) when a new domain rule is required. Our solution meets this goal in two significant ways: (1) by allowing the system to make use of old data rather than requiring new data when learning is indicated; contextual heuristics guide the selection of relevant examples from a memory constructed by the system; (2) by developing a scheme for automatically *generating* the contextual heuristics required for learning.

Our work is significant not only as it is a solution to the problem of incompleteness, but more generally as it is a study of one type of domain-independent contextual knowledge that can be explicitly exploited by similarity-based learning, an otherwise ‘‘knowledge-poor’’ technique. We have verified that our method is successful by implementing it in Gemini and testing it in four domains. Gemini is among the first learning systems to successfully integrate EBL and SBL.

The following section reviews the problems addressed in greater detail, discussing our solution to each. In Section 7.3 we reflect on the work, describing its limitations and our future directions. We conclude with a summary of our contributions.

7.2. Review of the Problems and Solutions

Our research addresses the important problem of domain theory incompleteness. Most knowledge-based systems are not designed to handle imperfect information. In particular, this problem occurs in explanation-based learning. Our research focuses on the incomplete domain theory problem in the context of EBL, though it can be used for rule-based systems in general.

There are a number of driving forces behind the solution we devised and implemented. These constitute a set of secondary problems, and are as follows:

1. The need to minimize assumptions about any additional knowledge used by a system.
2. The problem of inductively learning without large data sets *and* with little human guidance.
3. The problem of making explicit to an inductive learning system some of the information that might otherwise be implicitly encoded into the system's procedural components. (That is, making part of the inductive bias explicit.)
4. Devising a testing methodology for solutions to the incomplete theory problem.

Machine learning serves many purposes. However, one important goal is to make systems able to function with minimal external intervention. The first of our secondary problems reflects this goal. One difficulty with previous solutions to domain theory incompleteness is that they have required auxiliary theories that might be incomplete as well. The need to carefully craft not just a first theory but an additional one only makes the task of a system implementer more complicated. Our desire was to minimize the burdens placed on a system writer. Thus our solution is based on similarity-based learning, which requires essentially no explicit domain knowledge. Unlike many inductive problems, however, ours focuses on learning in the context of an existing (partial) knowledge base. Thus we devised a mechanism by which contextual knowledge can guide the induction. We apply a set of contextual heuristics. To ensure that a system implementer is not required to hand-code these heuristics, we present an algorithm that generates them from an itemization of available contextual knowledge.

The second of the above problems also relates to the goal of system autonomy. Though little explicit knowledge is required by induction, examples are needed. From these, the SBL system extrapolates new rules. A difficulty arises if large numbers of examples are not readily available. Furthermore, input examples are often explicitly labeled with their class membership. An external source may not always be available to perform the task of labeling. Our contextual heuristics help to overcome both of these problems. They use explanatory knowledge to label examples as positive or negative relative to a given concept to be learned. They can also generalize a new rule based upon explanatory information alone. We present an algorithm for storing examples input to a system, so that they can be efficiently retrieved when learning is required.

As already mentioned, our solution is largely based on induction, or similarity-based learning, as that requires relatively little explicitly encoded domain knowledge. A problem with SBL, however, is that knowledge is often implicitly encoded within the procedures of the method. Our goal was to create a mechanism by which some of that knowledge could be maintained and used explicitly. Contextual heuristics fill this need.

Finally, the solution to any problem requires verification. One difficulty that needed to be addressed was that of devising a general methodology for testing solutions to the problem of domain theory incompleteness. We present a method for evaluating learned rules based upon their similarity to expert-encoded rules. This ensures not only classification accuracy of learned knowledge, but conceptual accuracy as well.

7.3. Reflections and Further Work

This research provides a solution to the important problem of domain theory incompleteness under conditions where interaction with the environment cannot be assumed or expected. Though our solution is widely applicable, as shown by the testing we performed in four domains, it is not without its limitations. In this section, we discuss both limitations of our approach as well as directions for future research.

7.3.1. Limitations

7.3.1.1. Restrictions on Domains

Gemini has been applied to four domains, as discussed earlier. Though the domains are diverse with respect to the complexity of their input representations, they are restricted. We assume that they are tractable and correct (i.e., never overly general).

They contain rules about classification and causality. However, they do not contain more complex relationships among entities in the domain, such as time-varying information. At present, Gemini would not be capable of learning rules missing from such a theory. One might imagine, however, that Gemini should be able to determine the input features involved in making a time-dependent inference. That is, without significant extension, it should at least be able to determine the features *relevant* to a rule, if not the specific relationships among those features. Even in a time-dependent domain Gemini's heuristics would be applicable.

7.3.1.2. Selection of Partial Explanations

The problem of selecting a ''best'' partial explanation was not a focus of this research. We have assumed that the problem of abduction is solvable. In order to mimic the automatic generation of good partial explanations, we have removed rules (by hand) from complete explanations in order to treat them as partial for induction.

The exception to this limitation is that Gemini is able to construct and select a partial explanation when a subgoal of the explanation cannot be inferred in *any* way using the domain theory. That is, a subgoal is found that must be proven, but there is no rule in the domain theory in which the subgoal is the right-hand-side, or conclusion.

7.3.1.3. Multiple Faults as Many Single Faults

Unlike approaches such as Rajamoney's and Wilkins', Gemini is not restricted to assuming that only a single rule is missing from any given explanation. However, only a single rule may be learned at a time. Thus each missing rule must be learned separately. In these cases, the partial explanation is treated the same as it would be if only one rule were missing. That is, it is assumed to contain sufficient information to focus induction in a reasonable manner. In future work it would be useful to consider clusters of rules that might be missing together.

7.3.1.4. No Retraction

Currently Gemini has no mechanism for retracting learned rules, even if incorrect. One way to avoid this problem would be to defer the incorporation of a new rule into memory until that rule seemed stable.

7.3.1.5. Searching for Useful Examples

In order to determine that a previously seen example meets the condition for applying a particular heuristic, every example in memory must be matched against the condition. This brute-force matching method is clearly inefficient. The organization of the GBM could be used to make this more efficient.

7.3.2. Extensions

Though Gemini is not without limitations, it has proven effective in solving the incomplete theory problem under assumptions less restrictive than earlier methods. In addition, we have identified other ways in which Gemini could be useful. This section discusses ways in which Gemini could be extended.

7.3.2.1. Using Gemini to Overcome Intractability

Our work has essentially ignored the issue of domain theory intractability. Gemini could, however, be useful in overcoming that problem as well. If a complete explanation cannot be derived in a timely manner, one might imagine stopping explanation generation after some ‘‘reasonable’’ period of time. The explanation could be treated as a partial one by Gemini. Gemini might then be able to create a single new rule that would, in effect, summarize or approximate the entire missing piece of the explanation.

7.3.2.2. Gemini as a Knowledge Engineering Tool

We have concentrated on minimizing Gemini’s need for interaction with a human. The system could be used, however, as a tool for knowledge engineering. Say that a knowledge base has been engineered by an expert. Rules from the knowledge base can be removed, and Gemini can be asked to re-create them based upon data it sees. The expert can then compare his/her rules against those created by Gemini. Though the knowledge base might be good in general, Gemini might find some rule to better express a particular concept.

A knowledge engineer might also interact with Gemini to find gaps in the knowledge base that had been missed. Not only would the presence of a gap be indicated, but Gemini could suggest the missing rule.

7.3.2.3. More Sophisticated Contexts

Contextual heuristics encoded in Gemini have been successful in finding rules missing from a domain theory. The heuristics make use of general information available in explanations, such as goal and intermediate concepts, and rules used. Other information could potentially be exploited by the system, however. For instance, information characterizing the structure of the explanation might be useful in further defining context. It is possible, however, that the time required to extract and process additional information would be too costly relative to its benefits. An important task will be to carry out further experiments to select the most informative and cost-effective contextual features.

7.4. Contributions to Machine Learning

This research makes contributions to two major areas of machine learning: explanation-based and similarity-based learning.

7.4.1. Explanation-based learning

Explanation-based learning, like AI systems in general, assumes that it initially has a perfect domain theory. This is clearly unrealistic in all but the simplest domains.

We have devised and implemented a solution to the problem of domain theory incompleteness. Our solution relaxes significant restrictive assumptions made by previous solutions:

- It does not require interaction with an expert, except when generating partial explanations.
- It does not require an auxiliary domain theory. The only additional information used is a set of contextual heuristics that:

  - need not be perfect;
  - can be generated automatically.
- It learns rules for the concepts actually missing, rather than simply being concerned with the accuracy of the augmented domain theory. Thus it maintains conceptual integrity.

7.4.2. Similarity-based learning

We use similarity-based learning as a basis for our solution to the problem of domain theory incompleteness. It is useful as it requires essentially no explicit domain knowledge. It does require large numbers of examples, however. These must not only be readily available, but they must often be labeled with their class membership. Furthermore, implicit information is often present in an SBL system, but is difficult to evaluate because of its implicit nature.

Our primary contribution to SBL is the introduction of contextual heuristics that automatically extract and make use of information available in the EBL system. Contextual heuristics are important because:

- They label examples as positive or negative members of a given class. Thus labeling need not be performed by an external source.
- They generalize learned rules based on explanatory information, rather than additional examples. Thus the number of examples required for induction is reduced.
- Their influence on learning can be explicitly evaluated.

7.4.3. General Contributions

Though our work focuses on learning rules for EBL domain theories, the work applies more generally to any knowledge-based system. The network fault domain theory used in testing this work is, in fact, taken from an expert system. Finally, Gemini, which is the implementation of this work, is one of the first systems to successfully integrated explanation-based and similarity-based learning techniques for the mutual benefit of the two.

Bibliography

[Araya 84] Araya, A. A.
Learning Problem Classes by Means of Experimentation and Generalization.
In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 11 - 15. Austin, Texas, 1984.

[Bareiss and Porter 87]
Bareiss, E. R. and Porter, B. W.
Protos: An Exemplar-Based Learning Apprentice.
In *Proceedings of the Fourth International Machine Learning Workshop*, pages 12 - 23. Irvine, California, 1987.

[Bergadano, et al. 87]
Bergadano, F., Giordana, A., and Saitta, L.
Integrating EBL and SBL Approaches to Knowledge Base Refinement.
Methodologies for Intelligent Systems.
Elsevier Science Publishing Co., Inc., 1987.

[Buchanan and Shortliffe 84]
Buchanan, B. G. and Shortliffe, E. H.
Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.
Addison-Wesley, Reading, Massachusetts, 1984.

[Building Expert Systems 83]
F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, eds.
Building Expert Systems.
Addison-Wesley, Reading, Massachusetts, 1983.

[Burstein 86] Burstein, M. H.
Concept Formation by Incremental Analogical Reasoning and Debugging.
Machine Learning: An Artificial Intelligence Approach, Volume II.
Morgan Kaufmann, Los Altos, California, 1986.

[Carbonell 83a] Carbonell, J. G.
Derivational Analogy in Problem Solving and Knowledge Acquisition.
In *Proceedings of the Second International Machine Learning Workshop*, pages 12 - 18. Champaign-Urbana, Illinois, 1983.

[Carbonell 83b] Carbonell, J. C.
Learning by Analogy: Formulating and Generalizing Plans from Past Experience.
Machine Learning: An Artificial Intelligence Approach.
Morgan Kaufmann, Los Altos, California, 1983.

[Clancey 84] Clancey, W. J.
NEOMYCIN: reconfiguring a rule-based system with application to teaching.
Readings in Medical Artificial Intelligence.
Addison-Wesley, Reading, Massachusetts, 1984.

[Cohen 91] Cohen, W. W.
The Generality of Overgenerality.
In *Proceedings of the Eighth International Machine Learning Workshop*, pages 490 - 494. Northwestern University, 1991.

[Danyluk 87] Danyluk, A. P.
The Use of Explanations for Similarity-Based Learning.
In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 274 - 276. Milan, Italy, 1987.

[Davis 76] Davis, R.
Applications of Meta-level Knowledge to the Construction, Maintenance, and Use of Large Knowledge Bases.
PhD thesis, Stanford University Department of Computer Science, 1976.

[De Raedt and Bruynooghe 89]
De Raedt, L. and Bruynooghe, M.
Towards Friendly Concept-Learners.
In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 849 - 854. Detroit, Michigan, 1989.

[DeJong 81] DeJong, G. F.
Generalizations Based on Explanations.
In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 67 - 69. Vancouver, B. C., Canada, 1981.

[DeJong 83] DeJong, G. F.
Acquiring Schemata through Understanding and Generalizing Plans.
In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*. Karlsruhe, West Germany, 1983.

[DeJong and Mooney 86]
DeJong, G. and Mooney, R.
Explanation-Based Learning: An Alternative View.
Machine Learning 1(2):145 - 176, 1986.

[Dietterich and Flann 88]
Dietterich, T. G. and Flann, N. S.
An Inductive Approach to Solving the Imperfect Theory Problem.
In *Proceedings of the AAAI Symposium on Explanation-Based Learning*, pages 42 - 46. Stanford University, 1988.
Unpublished Proceedings.

[Ellman 87] Ellman, T.
Explanation-Based Methods for Simplifying Intractable Theories.
Technical Report CUCS-265-87, Columbia University Department of Computer Science, 1987.

[Ellman 88a] Ellman, T.
Explanation-Directed Search for Simplifying Assumptions.
In *Proceedings of the AAAI Symposium on Explanation-Based Learning*, pages 95 - 99. Stanford University, 1988.
Unpublished Proceedings.

[Ellman 88b] Ellman, T.
Approximate Theory Formation: An Explanation-Based Approach.
In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 570 - 574. St. Paul, Minnesota, 1988.

[Ellman 89a] Ellman, T.
Explanation-Based Learning: A Survey of Programs and Perspectives.
Computing Surveys 21(2):163 - 221, 1989.

[Ellman 89b] Ellman, T.
Integrated Analytic and Empirical Learning of Approximations for Intractable Theories.
PhD thesis, Columbia University Department of Computer Science, 1989.

[Eshelman 88] Eshelman, L.
1988
Personal Communication.

[Fawcett 89] Fawcett, T.
Learning from Plausible Explanations.
In *Proceedings of the Sixth International Machine Learning Workshop*, pages 37 - 39. Cornell University, 1989.

[Fisher 87] Fisher, D. H.
Conceptual Clustering, Learning From Examples, and Inference.
In *Proceedings of the Fourth International Machine Learning Workshop*, pages 38 - 49. Irvine, California, 1987.

[Flann and Dietterich 86]
Flann, N. S. and Dietterich, T. G.
Selecting Appropriate Representations for Learning from Examples.
In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 460 - 466. Philadelphia, Pennsylvania, 1986.

[Forbus 84] Forbus, K.
Qualitative Process Theory.
Artificial Intelligence 24:85 - 168, 1984.

[Genest et al. 90] Genest, J., Matwin, S., Plante, B.
Explanation-Based Learning with Incomplete Theories: A Three-step Approach.
In *Proceedings of the Seventh International Conference on Machine Learning*, pages 286 - 294. Austin, Texas, 1990.

[Ginsberg 88] Ginsberg, A.
Theory Revision via Prior Operationalization.
In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 590 - 595. St. Paul, Minnesota, 1988.

[Hall 86] Hall, R. J.
Learning by Failing to Explain.
In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 568 - 572. Philadelphia, Pennsylvania, 1986.

[Hammond 88] Hammond, K.
Case-Based Planning: Viewing Planning as a Memory Task.
In *Proceedings of the Case-Based Reasoning Workshop*, pages 17 - 20. Clearwater Beach, Florida, 1988.

[Hirsh 89] Hirsh, H.
Incremental Version-Space Merging: A General Framework for Concept Learning.
PhD thesis, Stanford University Department of Computer Science, 1989.

[Kedar-Cabelli 85]
Kedar-Cabelli, S.
Purpose-Directed Analogy.
In *Proceedings of the Cognitive Science Society Conference*. Irvine, California, 1985.

[Kedar-Cabelli 88]
Kedar-Cabelli, S.
Formulating Concepts and Analogies According to Purpose.
PhD thesis, Rutgers University Department of Computer Science, 1988.

[Keller 87] Keller, R. M.
The Role of Explicit Contextual Knowledge in Learning Concepts to Improve Performance.
PhD thesis, Rutgers University Department of Computer Science, 1987.

[Kodratoff 87] Kodratoff, Y. and Tecuci, G.
DISCIPLE-1: Interactive Apprentice System in Weak Theory Fields.
In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 271 - 273. Milan, Italy, 1987.

[Kodratoff and Tecuci 87]
Kodratoff, Y. and Tecuci, G.
What is an Explanation in DISCIPLE?
In *Proceedings of the Fourth International Machine Learning Workshop*, pages 160 - 166. Irvine, California, 1987.

[Kolodner 83] Kolodner, J. L.
Maintaining Organization in a Dynamic Long-Term Memory.
Cognitive Science 7(4):243 - 280, 1983.

[Kolodner 87] Kolodner, J. L.
Extending Problem Solver Capabilities Through Case-Based Inference.
In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 167 - 178. Irvine, California, 1987.

[Laird et al. 90] Laird, J., Hucka, M., Yager, E., and Tuck, C.
Correcting and Extending Domain Knowledge using Outside Guidance.
In *Proceedings of the Seventh International Conference on Machine Learning*, pages 235 - 243. Austin, Texas, 1990.

[Lebowitz 80] Lebowitz, M.
Generalization and Memory in an Integrated Understanding System.
PhD thesis, Yale University Department of Computer Science, 1980.

[Lebowitz 86a] Lebowitz, M.
Integrated Learning: Controlling Explanation.
Cognitive Science 10:219 - 240, 1986.

[Lebowitz 86b] Lebowitz, M.
Concept Learning in a Rich Input Domain: Generalization-Based Memory.
Machine Learning: An Artificial Intelligence Approach, Volume II.
Morgan Kaufmann, Los Altos, California, 1986.

[Lebowitz 87] Lebowitz, M.
Experiments with Incremental Concept Formation: UNIMEM.
Machine Learning 2(2):103 - 138, 1987.

[Lebowitz 90] Lebowitz, M.
The Utility of Similarity-based Learning in a World Needing Explanation.
Machine Learning: An Artificial Intelligence Approach, Volume III.
Morgan Kaufmann, San Mateo, California, 1990.

[Machine Learning, Volume I 83]
Michalski, R. S., Carbonell, J. G.,and Mitchell, T. M.
Machine Learning: An Artificial Intelligence Approach.
Morgan Kaufmann, Los Altos, California, 1983.

[Michalski and Ko 88]
Michalski, R. S. and Ko, H.
On the Nature of Explanation or Why Did the Bottle Shatter?
In *Proceedings of the AAAI Symposium on Explanation-Based Learning*, pages 12 - 16. Stanford University, 1988.
Unpublished Proceedings.

[Michalski and Stepp 83]
Michalski, R. S. and Stepp, R. E.
Automated Construction of Classifications: Conceptual Clustering Versus Numerical Taxonomy.
IEEE Transactions on Pattern Analysis and Machine Intelligence 5(4):396 - 409, 1983.

[Minton 88] Minton, S.
Learning Effective Search Control Knowledge: An Explanation-Based Approach.
PhD thesis, Carnegie-Mellon University Department of Computer Science, 1988.

[Mitchell 78] Mitchell, T. M.
Version Spaces: An Approach to Concept Learning.
PhD thesis, Stanford University Department of Computer Science, 1978.

[Mitchell 80] Mitchell, T. M.
The Need for Biases in Learning Generalizations.
Technical Report CBM-TR-117, Rutgers University Department of Computer Science, 1980.

[Mitchell 84] Mitchell, T. M.
Toward Combining Empirical and Analytical Methods for Inferring Heuristics.
Human and Artificial Intelligence.
North Holland Publishing Company, Amsterdam, 1984.

[Mitchell et al. 86]
Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S.
Explanation-Based Generalization: A Unifying View.
Machine Learning 1(1):47 - 80, 1986.

[Ng and Mooney 90]
Ng, H. T. and Mooney, R. J.
The Role of Coherence in Constructing and Evaluating Abductive Explanations.
In *Proceedings of the AAAI Symposium on Automated Abduction*. Stanford University, 1990.
Unpublished Proceedings.

[O'Rorke 90] O'Rorke, P.
Working Notes of the AAAI Symposium on Automated Abduction.
Technical Report TR 90-32, University of California, Irvine, 1990.

[Ourston and Mooney 90]
Ourston, D. and Mooney, R.
Changing the Rules: A Comprehensive Approach to Theory Refinement.
In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 815 - 820. Boston, Massachusetts, 1990.

[Pazzani 87] Pazzani, M. J.
Inducing Causal and Social Theories: A Prerequisite for Explanation-Based Learning.
In *Proceedings of the Fourth International Machine Learning Workshop*, pages 230 - 241. Irvine, California, 1987.

[Pazzani 88] Pazzani, M. J.
Learning Causal Relationships: An Integration of Empirical and Explanation-Based Learning Methods.
PhD thesis, UCLA Department of Computer Science, 1988.

[Pazzani et al. 86] Pazzani, M., Dyer, M., and Flowers, M.
The Role of Prior Causal Theories in Generalization.
In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 545 - 550. Philadelphia, Pennsylvania, 1986.

[Pazzani et al. 87] Pazzani, M., Dyer, M., and Flowers, M.
Using Prior Learning to Facilitate the Learning of New Causal Theories.
In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 277 - 279. Milan, Italy, 1987.

[Quinlan 86] Quinlan, J. R.
Induction of Decision Trees.
Machine Learning 1(1):81 - 106, 1986.

[Radio Manual 86]
Department of the Army.
Technical Manual - Unit Maintenance.
Technical Report TM 11-5820-890-20-1, Department of the Army, 1986.

[Rajamoney 88] Rajamoney, S. A.
Experimentation-Based Theory Revision.
In *Proceedings of the AAAI Symposium on Explanation-Based Learning*, pages 7 - 11. Stanford University, 1988.
Unpublished Proceedings.

[Rajamoney 89] Rajamoney, S. A.
Explanation-based theory revision: An approach to the problems of incomplete and incorrect theories.
PhD thesis, University of Illinois, Computer Science Department, 1989.

[Rajamoney and DeJong 87]
Rajamoney, S. and DeJong, G.
The Classification, Detection and Handling of Imperfect Theory Problems.
In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 205 - 207. Milan, Italy, 1987.

[Rajamoney et al. 85]
Rajamoney, S., DeJong, G., and Faltings, B.
Towards a Model of Conceptual Knowledge Acquisition Through Directed Experimentation.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 688 - 690. Los Angeles, California, 1985.

[Russell 86] Russell, S. J.
Analogical and Inductive Reasoning.
PhD thesis, Stanford University Department of Computer Science, 1986.

[Salzberg 83] Salzberg, S.
Generating Hypotheses to Explain Prediction Failures.
In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 352 - 355. Washington, DC, 1983.

[Salzberg 85] Salzberg, S.
Heuristics for Inductive Learning.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 603 - 609. Los Angeles, California, 1985.

[Silver 86] Silver, B.
Precondition Analysis: Learning Control Information.
Machine Learning: An Artificial Intelligence Approach, Volume II.
Morgan Kaufmann, Los Altos, California, 1986.

[Silver 88] Silver, B.
A Hybrid Approach in an Imperfect Domain.
In *Proceedings of the AAAI Symposium on Explanation-Based Learning*, pages 52 - 56. Stanford University, 1988.
Unpublished Proceedings.

[Smith et al. 85] Smith, R., Mitchell, T., Winston, H., and Buchanan, B.
Representation and Use of Explicit Justifications for Knowledge Base Refinement.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 673 - 680. Los Angeles, California, 1985.

[Stepp and Michalski 86]
Stepp, R. E. and Michalski, R. S.
Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects.
Machine Learning: An Artificial Intelligence Approach, Volume II.
Morgan Kaufmann, Los Altos, California, 1986.

[Swaminathan 88] Swaminathan, K.
Integrated Learning with an Incomplete and Intractable Domain Theory: The Problem of Epidemiological Diagnosis.
In *Proceedings of the AAAI Symposium on Explanation-Based Learning*, pages 62 - 66. Stanford University, 1988.
Unpublished Proceedings.

[Utgoff 86] Utgoff, P. E.
Shift of Bias for Inductive Concept Learning.
Machine Learning: An Artificial Intelligence Approach, Volume II.
Morgan Kaufmann, Los Altos, California, 1986.

[Utgoff and Saxena 88]
Utgoff, P. E., and Saxena, S.
Obtaining Efficient Classifiers from Explanations.
In *Proceedings of the AAAI Symposium on Explanation-Based Learning*, pages 47 - 51. Stanford University, 1988.
Unpublished Proceedings.

[Van Lehn 87] Van Lehn, K.
Learning One Subprocedure per Lesson.
Artificial Intelligence 31:1 - 40, 1987.

[Watanabe and Elio 87]
Watanabe, L. and Elio, R.
Guiding Constructive Induction for Incremental Learning from Examples.
In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 293 - 296. Milan, Italy, 1987.

[Wilkins 88] Wilkins, D. C.
Knowledge Base Refinement Using Apprenticeship Learning Techniques.
In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 646 - 651. St. Paul, Minnesota, 1988.

[Winston 72] Winston, P. H.
Learning Structural Descriptions from Examples.
The Psychology of Computer Vision.
McGraw-Hill, New York, 1972.

[Winston et al. 83]
Winston, P. H., Binford, T. O., Katz, B., and Lowry, M.
Learning Physical Descriptions from Functional Definitions, Examples, and Precedents.
In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 433 - 439. Washington, DC, 1983.

Appendix A

Selected Heuristics for Rule Induction

This appendix lists a selected set of 94 heuristics for rule induction following a brief guide to reading the heuristics. The 94 are only a subset of the full set of heuristics that would be generated following the methodology given in Chapter 4. Section A.3 of this appendix explains the selection of this subset and describes the algorithm used in generating it.

A.1. A Guide to Reading Heuristics

The heuristics given here may be viewed as rules, with a condition for application, and an action to be taken if that condition holds. Chapter 4 described a method for deriving these heuristics from information available in an EBL setting. That information describes attributes of EBL input examples as well as their corresponding explanations.

Each heuristic given here is listed with a brief description of it. First, however, we define a number of terms used in the heuristics, discussing how those terms are interpreted by Gemini.

Reading Conditions on Heuristic Applicability

Heuristics that guide the construction of missing domain theory rules are themselves rule-like. A heuristic has a condition that describes situations to which it can be applied, as well as an action to be taken if appropriate. The conditional part of a heuristic is a conjunction of relationships that must hold between a current example and another example seen earlier by EBL. If the condition of a heuristic correctly describes the relationship between current example x and a previously seen example y, we say that y is *retrieved* from memory. As discussed in Chapter 4, relationships between the examples are described by the similarity or difference of their attributes. Attributes describe both input examples to EBL as well as their corresponding explanations.

Input example features are handled differently from attributes of explanations. Similarity or difference of input features is determined by considering specific feature *values*. The similarity or difference of individual values (and the frames comprised of them) are determined by the matching algorithms described in Chapter 3. When input example features are referenced within an explanation — as they are in the explanation leaves — conditions on heuristic applicability refer only to the feature *names*.

A conjunct of the complete condition may refer to the similarity or difference of an attribute, where the single attribute may be described by a set of values. For instance, a heuristic may require that the leaves of two explanations be the same. As indicated above, similarity or difference of input example frames (described by sets of features) is determined by the matching algorithms of Chapter 3. If a condition requires that two explanations be the same in a particular attribute, they must be *identical* in that attribute. For example, if a condition requires that two explanations share inferred subgoals, then *all* the inferred subgoals in the two must be identical. That is, the difference between the two sets of subgoals must be empty. The same holds for other explanation attributes, with the exception of missing subgoals. Since rule induction as implemented in Gemini concentrates on a single missing subgoal at any time, two explanations must only share the subgoal in question if ‘‘sameness’’ is required.

Reading Heuristic Actions

When it is determined that the conditions of a heuristic hold, a specified action is taken. A heuristic action works with the premise of the rule being constructed. The action might state explicitly that a set of features mentioned in that premise is to be kept (thus eliminating the remaining features). It might also state that a particular set of features is to be eliminated from the new rule premise. When an action refers to a feature, it refers to the pair of the feature’s *name* and *value*.

At any point if ‘‘sameness’’ is mentioned with respect to feature values, it does not mean absolute match but any reasonable similarity as defined in Chapter 3.

A.2. Selected Heuristics

1. IF

```
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is SAME-AS past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

This heuristic corresponds to finding the most specific generalization of positive examples of a concept to be learned. The example retrieved from memory in this case is an example of the concept - i.e., missing subgoal - to be learned. In addition, it is essentially identical to the initial example both in terms of its input description and its corresponding explanation. A conservative approach to finding the definition of the missing concept for the two finds the most specific set of attributes common to them.

2. IF

```
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

As in the previous heuristic, here the retrieved example is a positive instance of the concept to be learned (i.e., the missing subgoal). Although the example retrieved appears to be very similar to the one to be completed by the new rule, there is a difference in the way their features interact to come to the same conclusion. In fact, although the same features are referred to in the respective explanations, the values of those features may very well be different. Given this, one might attempt to keep the distinction between the examples by eliminating the possibility of the new rule referring to aspects of the retrieved *explanation*.

3. IF

```
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

The original example x, for which a new rule is needed, and the retrieved example y appear to be quite similar. The difference in the rules used to prove the two identical goal concepts is probably only minor and is likely the result of minor differences in a subset of feature values. As the respective missing subgoals are also different, they probably refer to the minor inconsistencies between x and y. Thus the features to concentrate on for the new rule should be those that distinguish x from the retrieved example y. Note the completely different interpretation of the conditions in this heuristic from those of the preceding one.

4.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

This heuristic is a slightly more general version of heuristic #1 that mimics learning from positive examples. Here the missing subgoal - if any at all - is not even mentioned in the heuristic. The assumption here is that examples x and y are so similar that y must be a positive example of the subgoal missing from x's explanation. (Given Gemini's implementation, that would, in fact, be true.)

5.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Although the example retrieved appears to be very similar to the one to be completed by the new rule, there is a difference in the way their features interact to come to the same conclusion. In fact, although the same features are referred to in the respective explanations, the values of those features may very well be different. Given this, one might attempt to keep the distinction between the examples by eliminating the possibility of the new rule referring to aspects of the retrieved explanation. Note that the conditions of this heuristic allow it to be applied to examples covered by heuristics #2 and #3. This heuristic performs the somewhat more conservative generalization of the new rule resulting from heuristic #2.

6.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     LEAVES in example y.
```

This heuristic assumes that the explanations of example x and the retrieved example y differ primarily in their ''lowest'' parts. That is, the explanations are probably quite similar close to their roots (or goals), but differ in the leaves as well as in the rules that connect the leaves to the rest of the explanations. One might assume that these slight differences arise from the fact that the rules used were slightly too specific and covered only some aspects of the relevant features for each case. That is, both explanations should have been more similar, but the rules used to generate them were too specific to cover all the relevant feature values. In order to force the two explanations to cover a more consistent set of features, the new rule for example x could mention those features referred to in y's explanation.

7. IF
```
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     SUBGOALS in example y.
```

Although the retrieved example y appears similar to the current example x, it is really quite different in the way the features interact to prove the identical final goal. In this case, assume that any features covered by the proven subgoals of y would already have been referred to if they were important to x's explanation. Eliminate them from the new rule needed to complete x.

8. IF
```
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     SUBGOALS in example y.
```

Although the retrieved example y appears similar to the current example x, it is really quite different in the way the features interact to prove the identical final goal. In this case, as in the previous one, assume that any features covered by the proven subgoals of y would already have been referred to if they were important to x's explanation. Eliminate them from the new rule needed to complete x. This is reasonable if the rule base is most likely very nearly complete. Note that the conditions of this heuristic allow it to apply to examples covered by both heuristic #6 and heuristic #7. Note also the vastly different ways that the conditions of #6 and #7 may be interpreted.

9. IF
```
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is SAME-AS past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
```

```
    EXAMPLE-FEATURES in example y.
```

This heuristic is a slightly more general version of heuristic #1 that performs learning by least generalization from positive examples. Here the leaves of the respective explanations are not mentioned in the heuristic. The assumption here is that examples x and y are so similar that the leaves of their explanations are probably identical as well. (Given Gemini's implementation, the leaf names would certainly be the same.)

10.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Here the retrieved example y appears similar to the current example x. Since the rules used were different, however, the features interact differently to prove the identical final goal. In this case, assume that any features covered by y's explanation would already have been referred to if they were important to x's explanation. Eliminate them from the new rule needed to complete x. This is reasonable if the rule base is most likely very nearly complete. Note that the conditions of this heuristic allow it to apply to examples covered by both heuristic #2 and heuristic #6. Note also the vastly different ways that the conditions of #2 and #6 may be interpreted.

11.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

The original example x, for which a new rule is needed, and the retrieved example y appear to be quite similar. The difference in the rules used to prove the two identical goal concepts is probably only minor and is likely the result of minor differences in a subset of feature values. As the respective missing subgoals are also different, they probably refer to the minor inconsistencies between x and y. Thus the features on which to concentrate for the new rule should be those that distinguish x from the retrieved example y. Note that the conditions of this heuristic allow it to be applied to cases covered by heuristic #3 and heuristic #7.

12.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
```

```
EXAMPLE-FEATURES in example y.
```

This heuristic is a more general version of heuristic #1 that performs learning by least generalization from positive examples. The assumption here is that examples x and y are so similar in the respects mentioned that they are probably nearly identical in all respects. (In fact, given Gemini's implementation, they would be.) Note that the conditions of this heuristic are more general (and therefore more widely applicable) than those of heuristics #4 and #9. This heuristic is **Test Heuristic 0** in the runs described in Chapter 5.

13.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     SUBGOALS in example y.
```

As in many earlier heuristics, although the retrieved example y appears similar to the current example x, it is somewhat different in the way the features interact to prove the identical final goal. In this case, assume that any features covered by the proven subgoals of y would already have been referred to if they were important to x's explanation. Eliminate them from the new rule needed to complete x. Note that the conditions of this heuristic are a generalization of heuristics #5, #8, #10, and #11.

14.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Here the retrieved example y is a positive example of the concept - i.e., missing subgoal - whose definition is to be learned for the current example x. Although the explanations for examples x and y differ somewhat, there are similarities between them as well. As they appear to have more similarities than they do differences, it might be expected that the rule proving the missing subgoal should be useful for both. Thus they are treated as positive examples of the missing subgoal. Furthermore, the same disjunct of the definition for that subgoal is assumed to correctly classify both cases.

15.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
```

```
      EXAMPLE-FEATURES in example y;
   ELIM features in rule that are the same as those covered by
      LEAVES in example y.
```

In this heuristic an extreme view is taken that, although there are similarities between x and y, the examples are really quite different. That is, all similarities are taken to be purely superficial and/or coincidental, rather than meaningful. In order to distinguish between the examples and to concentrate on their differences, all features common to the two are eliminated from the new rule to be created. Furthermore, the new rule will not refer to any features playing a role in the explanation for y. Given Gemini's implementation of heuristics, this second elimination step would be redundant.

16.
```
IF
   current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
   current example x is SAME-AS past example y in GOAL-CONCEPT AND
   current example x is DIFF-FROM past example y in SUBGOALS AND
   current example x is SAME-AS past example y in LEAVES AND
   current example x is DIFF-FROM past example y in RULES-USED
THEN
   ELIM features in rule that are the same as those covered by
      EXAMPLE-FEATURES in example y;
   ELIM features in rule that are the same as those covered by
      LEAVES in example y.
```

As in heuristic #15, the extreme view is taken here that, although there are similarities between x and y, the examples are really quite different. That is, all similarities are taken to be purely superficial and/or coincidental, rather than meaningful. In order to distinguish between the examples and to concentrate on their differences, all features common to the two are eliminated from the new rule to be created. Furthermore, the new rule will not refer to any features playing a role in the explanation for y. The conditions of this heuristic are a generalization of those in heuristics #14 and #15. Note the extremely different interpretations of the conditions as indicated in the actions to be taken.

17.
```
IF
   current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
   current example x is SAME-AS past example y in GOAL-CONCEPT AND
   current example x is DIFF-FROM past example y in SUBGOALS AND
   current example x is DIFF-FROM past example y in LEAVES AND
   current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
   current example x is DIFF-FROM past example y in RULES-USED
THEN
   KEEP features in rule that are the same as those covered by
      EXAMPLE-FEATURES in example y;
   ELIM features in rule that are the same as those covered by
      LEAVES in example y.
```

This heuristic takes the view that, although different, the retrieved example y is sufficiently similar to x that it should probably be covered by the same rule learned for x's missing subgoal. Thus the features shared by the two examples are the focus of rule induction. Furthermore, this heuristic assumes that all features relevant to y's explanation would already have appeared in x's if they were necessary there as well. Thus they are eliminated from the missing concept definition.

18. IF
```
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Although the frame description of the retrieved example y is similar to x's, and though they are members of the same goal concept, the features important for classifying each as a member of that concept are quite different (as indicated by the difference in their respective explanations). In order to follow this thread, this heuristic concentrates on the features of x that are different from those in y. This is **Test Heuristic 5** in the runs of Chapter 5.

19. IF
```
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

This heuristic assumes that disjunctive ways of proving the same final goal concept will be vastly different. In this case, it is sensible to remove all similarity to the retrieved example y from the rule to complete the explanation for x. Note that this is a modified version of the ‘‘disjuncts are exclusive’’ rule described in Chapters 3 and 4. Note also that the conditions of this heuristic are a generalization of the conditions of #17 and #18, both of which where interpreted in vastly different manners.

20. IF
```
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y;
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Although the retrieved example y differs somewhat from the current example x, it is also a positive example of the concept to be learned - i.e., the missing subgoal. Furthermore, y is quite similar to x in a number of other respects. Thus it is likely that the two examples should be covered by a single definition for the missing subgoal. As a result, their shared features are noted in the new rule. Additionally, features from y's explanation are deleted from the rule as they would probably have been mentioned in x's explanation already if they were important there. Note that the conditions of this rule are a generalization of those in #14 and #17.

21. IF
```
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

This heuristic assumes that disjunctive ways of proving the same final goal concept will be vastly different. In this case, it is sensible to remove all similarity to the retrieved example y from the rule to complete the explanation for x. Note that this is a modified version of the ‘‘disjuncts are exclusive’’ rule described in Chapters 3 and 4. Note also that the conditions of this heuristic are a generalization of the conditions of #15 and #18.

22. IF
```
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

This is a slightly modified version of the ‘‘disjuncts are exclusive’’ rule, as are heuristic #19 and heuristic #21. Note that the conditions of this heuristic are a generalization of the conditions of #16, #19, #20, and #21.

23. IF
```
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y;
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Although the retrieved example y differs somewhat from the current example x, it is also a positive example of the concept to be learned - i.e., the missing subgoal. Furthermore, y is also quite similar to x in a number of respects. Thus, it is likely that the two examples should be covered by a single definition for the missing subgoal. As a result, their shared features are noted in the new rule. Additionally, features from y’s explanation are deleted from the rule as they would probably have been mentioned in x’s explanation if they were important there.

24. IF
```
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
```

```
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Here the retrieved example y, although appearing similar to the current example x, is really quite different in its stated goal concept membership as well as in its explanation. It is therefore probably reasonable to treat it as a negative example of the missing subgoal to be learned. As a result attention is drawn to the features of x that differ from y.

25.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Here the retrieved example y, although appearing similar to the current example x, is really quite different in its stated goal concept membership as well as in its explanation. It is therefore probably reasonable to treat it as a negative example of the missing subgoal to be learned. As a result attention is drawn to the features of x that differ from y. Note that the conditions of this heuristic make it more generally applicable than #23 and #24, although its conditions are interpreted differently from #23.

26.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y;
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Although the retrieved example y differs from the current example x, it is also a positive example of the concept to be learned - i.e., the missing subgoal. Furthermore, y is also quite similar to x in its input representation. Thus, it is possible that the two examples should be covered by a single definition for the missing subgoal. As a result, their shared features are noted in the new rule. Additionally, features from y's explanation are deleted from the rule as they would probably have been mentioned in x's explanation if they were important there. Note that there is sufficient difference between x and y that the assumptions of this heuristic could be quite wrong. This is the same as **Test Heuristic 6** in the runs of Chapter 5.

27.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
```

```
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Here the retrieved example y, although appearing similar to the current example x in its input representation, is really very different in its stated goal concept membership as well as in its explanation. It is therefore reasonable to treat it as a negative example of the missing subgoal to be learned. As a result, attention is drawn to the features of x that differ from those in y. This is **Test Heuristic 7** studied in Chapter 5.

28.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though there is no explicit information in this heuristic about the relationship between x's missing subgoal and y, x and y are so different that it is reasonable to treat y as a negative example of the missing subgoal to be learned. Attention is drawn to the features of x that distinguish it from y. Note that the conditions of this heuristic make it more generally applicable than #26 and #27, though the conditions of #26 are interpreted very differently.

29.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Although the retrieved example y differs from the current example x, it is also a positive example of the concept to be learned - i.e., the missing subgoal. Furthermore, y is similar to x in its input representation. Thus it is possible that the two examples should be covered by a single definition for the missing subgoal. As a result, their shared features are noted in the new rule. Note that the conditions of this heuristic make it more generally applicable than #23 and #26. As noted in #26, the assumptions made in this heuristic could quite possibly cause it to fail.

30.
```
IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
```

```
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Here the retrieved example y, although similar to the current example x in its input representation, is really quite different in its stated goal concept membership as well as in its explanation. It is therefore reasonable to treat it as a negative example of the missing subgoal to be learned. Attention is drawn to the features of x that distinguish it from y. Note that the conditions of this heuristic make it more generally applicable than #24 and #27.

```
31. IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

As in the previous several heuristics, this one treats the retrieved example y as a negative example of the missing concept to be learned. Its conditions are a generalization of those in #25, #28, #29, and #30.

```
32. IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is SAME-AS past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Although the retrieved example y looks different from the current example x in its input representation, it is also a positive example of the concept to be learned - i.e., the missing subgoal. Furthermore, y is quite similar to x with respect to its complete explanation structure. Thus it is possible that the two examples differ in non-substantive ways and should be covered by a single definition for the missing subgoal. As a result, their shared features are noted in the new rule.

```
33. IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Although the retrieved example y differs from the current example x, it is also a positive example of the concept to be learned - i.e., the missing subgoal. Furthermore, y is quite similar to x with respect to its complete explanation structure. As in the previous heuristic, their shared features are noted in the new rule.

34. IF

```
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though the retrieved example y shares many attributes of the current example x, it differs from it in key ways. Key features of the two are sufficiently different that different explanations were generated to reach the same final goal. Thus attention is focused on those features distinguishing the two.

35. IF

```
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Note that the conditions of this heuristic are more general than those of #32. In fact, since our implementation does not allow missing subgoals to differ in examples where all rules applied are identical, this heuristic covers exactly those cases that #32 would.

36. IF

```
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Although the input representation of the retrieved example y differs from the current example x, it is quite similar to x with respect to its explanation structure. Thus it is possible that the two examples differ in non-substantive ways and should be covered by the definition of the missing subgoal. As a result, their shared features are noted in the new rule. Note that the conditions of this heuristic are more general than those of #33 and #34, though the conditions of #34 are treated quite differently.

37. IF

```
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y;
```

```
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

This heuristic takes the view that there is sufficient difference between examples x and y that they should be treated essentially as opposites in order to maximize that difference that already exists. As x and y are similar in key attributes such as concept membership, this is a good example of a particularly bad heuristic.

38.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Key features of the two examples here are sufficiently different that different explanations were generated to reach the same final goal. Thus attention is focused on those features distinguishing the two.

39.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

This heuristic takes the view that there is sufficient difference between examples x and y that they should be treated essentially as opposites in order to maximize the difference that already exists. As x and y are similar in key attributes such as concept membership, this may be a bad heuristic, though perhaps not quite as bad as #37. Note that the conditions of this heuristic are a generalization of the conditions of #37 and #38.

40.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is SAME-AS past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Although the input representations of x and y appear somewhat different, those aspects most relevant to their corresponding explanations are the same. It is possible that any dissimilarity between the two is limited to irrelevant and/or unimportant features. The two examples are treated as positive examples of the same disjunct in the definition of the missing concept. The conditions of this heuristic are a generalization of those in #32. Given the restriction of our implementation that similarity of rules necessarily implies similarity of leaves, the conditions of this heuristic just describe those of #32 more succinctly.

41.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

This heuristic takes the view that there is sufficient difference between examples x and y that they should be treated essentially as opposites in order to maximize the difference that already exists. As x and y are similar in key attributes such as concept membership, this may be a bad heuristic, though perhaps not quite as bad as #37. Note that the conditions of this heuristic are a generalization of the conditions of #33 and #37.

42.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

The retrieved example y is not obviously an example of the unproven subgoal in x's explanation. Furthermore, from other evidence distinguishing y from x, it is quite likely that y is, in fact, a negative example of that subgoal. As a result, y is treated as such, and the heuristic concentrates on features distinguishing x and y. Note that the conditions of this heuristic are a generalization of those in #34 and #38.

43.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Although the input representations of x and y appear somewhat different, those aspects most relevant to their corresponding explanations are the same. It is possible that any dissimilarity between the two is limited to irrelevant and/or unimportant features. The conditions of this heuristic are a generalization of those in #35 and #40. Given restrictions on conditions due to our implementation, this heuristic succinctly describes #32, of which #35 and #40 are generalizations.

44.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

The retrieved example y is not obviously an example of the unproven subgoal in x's explanation. Furthermore, from other evidence distinguishing y from x, it is possible that y is, in fact, a negative example of that subgoal. As a result, y is treated as such, and the heuristic concentrates on features distinguishing the two. Note that the conditions of this heuristic are a generalization of those in #36, #39, #41, and #42. Note also that there is enough similarity between examples x and y that this heuristic would conceivably fail often.

45.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though the retrieved example y is a positive example of the concept to be learned, it also differs from the input example x in a number of significant ways. In fact, though the features mentioned in the leaves of their explanations are the same, it is likely that the *values* of those features actually differ. Thus we might safely assume that the definition of the unproven subgoal should be disjunctive and that examples x and y belong to different disjuncts. As such, attention is drawn to those features that distinguish them. This may be problematic in that the two disjuncts might share some features while varying in others.

46.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

The retrieved example y differs in significant ways from the input example x. In addition to the difference in their input representations, their explanations showing membership in an identical goal concept are radically different. Attention is focused on features distinguishing the two.

47.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

As in the previous case, here the retrieved example y differs in significant ways from the input example x. In addition to the difference in their input representations, their explanations showing membership in an identical goal concept are quite different. Attention is focused on features distinguishing the two examples. Note that the conditions of this heuristic are a generalization of those in #45 and #46.

48.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though the retrieved example y is a positive example of the concept to be learned, it also differs from the input example x in a number of significant ways. As a result, attention is drawn to the features distinguishing x from y.

49.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though both x and y are examples of an identical goal concept, both are also quite different. Attention is drawn to the features distinguishing them. This is an instance of the ‘‘disjuncts are exclusive’’ idea described earlier.

50.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

As in the previous heuristic, the retrieved example y is sufficiently different from the input example x that attention is focused on features distinguishing the two examples. Note that the conditions of this heuristic make it more generally applicable than #48 and #49.

51.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though the retrieved example y is a positive example of the concept to be learned, it also differs from the input example x in a number of significant ways. Thus we might safely assume that the definition of the unproven subgoal should be disjunctive and that examples x and y belong to different disjuncts. As such, attention is drawn to those features that distinguish them. The conditions of this heuristic make it more generally applicable than #45 and #48. Potential problems of #45 would apply here as well.

52. IF
```
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though both x and y are examples of an identical goal concept, both are also quite different. Attention is given to the features distinguishing them. Note that the conditions of this heuristic are a generalization of those in #46 and #49.

53. IF
```
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

As in the previous several heuristics, examples x and y are sufficiently different here that any similarity between them is removed from the concept definition learned for x. Conditions of this rule are a generalization of those in #47, #50, #51, and #52.

54. IF
```
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though the retrieved example y is a positive example of the concept to be learned, it also differs from the input example x in a number of significant ways. Thus we might safely assume that the definition of the unproven subgoal should be disjunctive and that examples x and y belong to different disjuncts. As such, attention is drawn to those features that distinguish them. This may be problematic in that the two disjuncts might share one set of features while varying in others.

55. IF

```
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

The retrieved example y is so radically different from the current example x, that attention is focused on their differing features in order to preserve the difference in their explanations.

56. IF

```
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though y's similarity or difference with respect to the missing subgoal is not indicated here, the example is sufficiently different from the current one that it is taken to be a negative example of the missing subgoal. The conditions of this heuristic are a generalization of those in #54 and #55.

57. IF

```
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though the retrieved example y is a positive example of the concept to be learned, it also differs from the input example x quite significantly. Thus we might safely assume that the definition of the unproven subgoal should be disjunctive and that examples x and y belong to different disjuncts. As a result, attention is drawn to those features that distinguish them. As described earlier, this may be problematic in that the two disjuncts might share one set of features while varying in others.

58. IF

```
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

No noted similarity appears to exist between the current example x and the retrieved example y. As a result, y is taken to be a negative example of the subgoal to be learned.

59.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though the relationship between unproven subgoals in x and y is not indicated, the two examples are so very different that y is taken to be a negative example of the subgoal missing for the current example x. Conditions of this heuristic are a generalization of the two preceding it.

60.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though the retrieved example y is a positive example of the concept to be learned, it also differs from the input example x quite significantly. Thus we might safely assume that the definition of the unproven subgoal should be disjunctive and that examples x and y belong to different disjuncts. As a result, attention is drawn to those features that distinguish them. As described earlier, this may be problematic in that the two disjuncts might share some features while varying in others. Note that the conditions of this heuristic are a generalization of those in #54 and #57.

61.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

As the retrieved example y differs from x in every specified attribute, it is taken to be a negative example of the unproven subgoal. Note that the conditions of this heuristic make it more generally applicable than #55 and #58.

62.
```
IF
  current example x is DIFF-FROM past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
```

```
EXAMPLE-FEATURES in example y.
```

Though the relationship between unproven subgoals in x and y is not indicated, the two examples differ in each of the attributes specified. As a result, y is taken to be a negative example of the subgoal to be learned for x. Conditions of this heuristic are a generalization of those in #56, #59, #60, and #61.

63.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is SAME-AS past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Here the retrieved example y not only requires the same subgoal missing for example x, but is similar to x in a variety of other ways as well. This heuristic creates a most specific generalization of the examples in building the left-hand-side of the missing rule. Note that the conditions of this heuristic are a generalization of those in #1 and #32.

64.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

The retrieved example y is clearly an example of the subgoal missing for the current example x. It is also similar to it in a variety of other respects. As in the previous heuristic, the least generalization of the two will be the basis for the definition of the missing concept. Due to the manner in which heuristics are implemented in Gemini, this might not be as good a heuristic as it appears. Stating that the leaves of the explanations are the same simply implies that the same features were referred to - it does not guarantee that the feature values were the same. Thus the examples x and y might be a bit more different than the condition here implies. Note that the conditions are a generalization of those in #2 and #33.

65.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

In this case x and y are quite similar, but have a number of key differences as well. Most specifically, they do not require the same subgoal to complete their respective proofs. If the shared leaves of the two explanations were necessary for completing them, it is likely that the two explanations would appear more similar - in fact, they might be missing the same subgoal. Thus it is assumed that they are not necessary. Note that the conditions of this heuristic make it more generally applicable than #3 and #34.

66.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Although it is not indicated that the examples x and y are missing the same subgoal, the implementation of Gemini rules ensures that that is the case. Regardless of the implementation, however, the examples are sufficiently similar that this might be assumed. Note that the conditions are a generalization of those in #4, #35, and #63.

67.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

In this case x and y are quite similar, but have a key difference as well. Thus y is taken to be a negative example of the subgoal to be learned for x. A somewhat conservative approach is taken in which the new rule for x does not necessarily eliminate all similarity between the two examples. Only a subset of similar features are eliminated. Note that the conditions of this heuristic make it more generally applicable than #5, #36, #64, and #65.

68.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

In this case the retrieved example y is a positive instance of the concept to be learned - i.e., the missing subgoal in x. Though y is somewhat different from x, it is treated as an instance to be covered by the same part of the disjunctive definition of the missing subgoal. Note the generalization of the conditions here from #6 and #37.

69.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
```

```
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

The retrieved example y is different from x with respect to its missing subgoal in spite of the fact that both examples are instances of the same goal concept. They differ in a number of other attributes as well. Thus y is taken to be a negative example of the missing subgoal and the features common to it and to x are eliminated from the definition being created. Note the generalization of conditions here from #7 and #38.

70.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though it is not clear whether x and the retrieved example y are missing the same subgoal, they differ in enough key attributes that y is assumed to be a negative example of that subgoal. Note the generalization of conditions here from #8, #39, #68, and #69.

71.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is SAME-AS past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Not only is y a positive example of the missing subgoal for x, but the two examples are similar in all attributes specified. (Due to Gemini's implementation, the attribute of leaves, not specified here, could be assumed the same in the two examples as well.) Thus they are taken to be examples that should be covered by a single definition for that missing subgoal. Note the generalization of conditions here from #9, #40, and #63.

72.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

In addition to sharing the same missing subgoal as the current example, the retrieved example y is similar in a number of other attributes as well. It is therefore taken to be a positive example not only of that missing subgoal, but of the same part of the definition of that subgoal. Note the generalization of conditions from those of #10, #41, #64, #68.

73.
```
IF
```

```
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

The retrieved example y is different from x with respect to its missing subgoal in spite of the fact that both examples are instances of the same goal concept. They differ in another specified attribute as well. Thus y is taken to be a negative example of the missing subgoal and the features common to it and to x are eliminated from the definition being created. Note the generalization of conditions here from #11, #42, #65, and #69.

74. IF
```
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is SAME-AS past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Although it is not indicated that the examples x and y are missing the same subgoal, the implementation of Gemini rules ensures that that is the case when all rules used are the same. Regardless of the implementation, however, the examples are sufficiently similar that this might be assumed. Note that the conditions here are a generalization of those in #12, #43, #66, and #71.

75. IF
```
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

In addition to sharing the same missing subgoal as the current example, the retrieved example y is similar in another important attribute as well. It is therefore taken to be a positive example not only of x's missing subgoal, but of the same part of the definition of that subgoal. Note that the potential difference between the examples in their other attributes might make this a bad heuristic to apply. Note also that the conditions of this heuristic are a generalization of those in #13, #44, #67, #70, #72, and #73.

76. IF
```
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y;
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Although both x and y require the same missing subgoal to complete proofs of membership in the same final goal concept, their proofs are also quite different. In this case, it is assumed that they are sufficiently different that a maximization of that difference is the correct approach to completing the explanations. This is a variation on the ‘‘disjuncts are different’’ heuristic described earlier. Note that examples x and y are sufficiently similar that this would almost certainly be a bad heuristic. Note also that the conditions of this heuristic make it more generally applicable than #14 and #45.

77.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

The retrieved example y is different from x with respect to its missing subgoal in spite of the fact that both examples are instances of the same goal concept. They differ in other attributes as well. Thus y is taken to be a negative example of the missing subgoal and the features common to it and to x are eliminated from the definition being created. Note the generalization of conditions here from #15 and #46.

78.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though the relationship between x and y with respect to the missing subgoal is not specified, the two examples are sufficiently different that y is assumed to be a negative example of the subgoal to be learned for x. Note that there may not be sufficient evidence of difference to make this a reasonable heuristic to apply. Note also that the conditions here are a generalization of those in #16, #47, #76, and #77.

79.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Though the retrieved example y looks quite different from the current example x, it is a positive example of x’s missing subgoal. It is taken to be an example that should be covered by a different disjunct of that concept than x is. In order to keep that distinction visible in the definition to complete x, features relating x and y are eliminated from the definition. A somewhat conservative approach is taken, however, eliminating only a subset of their shared features. Note that the conditions of this heuristic are a generalization of those in #17 and #48. This heuristic is the same as **Test Heuristic 8**, for which test runs were described in Chapter 5.

80.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Though the retrieved example y looks quite different from the current example x, it is an example of the same final goal concept. Therefore, though it might be reasonable to treat y as a negative example of the concept to be learned and to eliminate from that definition all features common to x and y, a somewhat conservative approach is taken in which only features appearing in the leaves of y's explanation are eliminated. Note that the conditions of this heuristic are a generalization of those in #18 and #49.

81.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Though the retrieved example y looks quite different from the current example x, it is an example of the same final goal concept. In fact, it is possible that it is a positive example of the missing subgoal as well. Therefore, though it might be reasonable to treat y as a negative example of the concept to be learned and to eliminate from that definition all features common to x and y, a somewhat conservative approach is taken in which only features appearing in the leaves of y's explanation are eliminated. Note that the conditions of this heuristic are a generalization of those in #19, #50, #79, and #80.

82.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Though the retrieved example y looks somewhat different from the current example x, it is a positive example of x's missing subgoal. It is taken to be an example that should be covered by a different disjunct of that concept than x. In order to keep that distinction visible in the definition to complete x, features relating x and y are eliminated from the definition. A somewhat conservative approach is taken, however, eliminating only a subset of their shared features. Note that the conditions of this heuristic are a generalization of those in #20, #51, #76, and #79.

83.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Though the retrieved example y looks quite different from the current example x, it is an example of the same final goal concept. Therefore, though it might be reasonable to treat y as a negative example of the concept to be learned and to eliminate from that definition all features common to x and y, a somewhat conservative approach is taken in which only features appearing in the leaves of y's explanation are eliminated. Note that the conditions of this heuristic are a generalization of those in #21, #52, #77, and #80.

84.
```
IF
  current example x is SAME-AS past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

As in the previous heuristic, though the retrieved example y looks somewhat different from the current example x, it is an example of the same final goal concept. Therefore, a somewhat conservative approach is taken in which only features appearing in the leaves of y's explanation are eliminated from the definition being learned. Note that the conditions of this heuristic are a generalization of those in #22, #53, #78, #81, #82, and #83.

85.
```
IF
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Though the retrieved example y is a positive example of the concept to be learned - the missing subgoal for x - it also differs from x in a number of respects. As a result, it is taken to belong to a different disjunct in a probably disjunctive definition of the subgoal. In order to reflect this distinction, the features common to x and y are eliminated from the definition to be used to complete x's explanation. Note that this heuristic is likely to fail as at least some subset of these might be necessary to partially describe the missing concept. Note also that the conditions of this heuristic are a generalization of those in #23 and #54.

86.
```
IF
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

In this, x and y differ quite substantially. Not only are they members of different final goal concepts, but their explanations differ in other respects as well. Therefore, y is taken to be a negative example of the subgoal missing from x's explanation and a definition for that subgoal concentrates on features distinguishing the examples. Note that the conditions of this heuristic are a generalization of those in #24 and #55.

87.
```
IF
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

As in the previous case, x and y differ quite a bit here. Not only are they members of different final goal concepts, but their explanations differ in other respects as well. Therefore, y is taken to be a negative example of the subgoal missing from x's explanation and a definition for that subgoal concentrates on features distinguishing the examples. Note that the conditions of this heuristic are a generalization of those in #25, #56, #85, and #86.

88.
```
IF
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Though the retrieved example y is a positive example of the subgoal missing from x's explanation, it also differs from x quite substantially. Therefore, it is reasonable to assume that the definition of the missing subgoal is disjunctive and that the two examples are covered by separate disjuncts. As a result, this heuristic concentrates on features that distinguish the two examples in order to find the definition that applies to x. Rather than finding the maximal difference between the examples, however, the only similarity explicitly eliminated is that defined by y's leaves. Note that the conditions of this heuristic are a generalization of those in #26 and #57.

89.
```
IF
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
```

```
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

In this case, the retrieved example y appears to have essentially no similarity to the current example x. As a result, the heuristic concentrates on maintaining the distinction by further focusing on features in which the two differ. Note that the conditions here are a generalization of those in #27 and #58.

90. IF
```
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

As in the previous heuristic, the retrieved example y appears to have essentially no similarity to the current example x. As a result, though there is no information with respect to their missing subgoals (if any), the heuristic concentrates on maintaining the distinction by further focusing on features in which the two differ. Note that the conditions here are a generalization of those in #28, #59, #88, and #89.

91. IF
```
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     LEAVES in example y.
```

Though the retrieved example y is a positive example of the subgoal missing from x's explanation, it also differs from x quite substantially. Therefore, it is reasonable to assume that the definition of the missing subgoal is disjunctive and that the two examples are covered by separate disjuncts. As a result, this heuristic concentrates on features that distinguish the two examples in order to find the definition that applies to x. Rather than finding the maximal difference between the examples, however, the only similarity explicitly eliminated is that defined by y's leaves. Note that the conditions of this heuristic are a generalization of those in #29, #60, #85, and #88.

92. IF
```
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

In this case, the retrieved example y appears to have essentially no similarity to the current example x. As a result, the heuristic concentrates on maintaining the distinction by further focusing on features in which the two differ. Note that the conditions here are a generalization of those in #30, #61, #86, and #89.

93.
```
IF
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

As in the previous heuristic, the retrieved example y appears to have essentially no similarity to the current example x. As a result, though there is no information with respect to their missing subgoals (if any), the heuristic concentrates on maintaining the distinction by further focusing on features in which the two differ. Note that the conditions here are a generalization of those in #31, #62, #87, #90, #91, and #92.

94.
```
ELIM features in rule that are covered by LEAVES in current
  example x.
```

This heuristic was used in the example trace of Chapter 4, and is the same as **Test Heuristic 9** in Chapter 5. It assumes that features of an input example will generally not be referred to multiple times within a single explanation of that example. As a result, all those features already referenced may safely be eliminated from the rule being constructed to complete the explanation.

A.3. How Selected Heuristics were Generated

This appendix lists 94 heuristics that might be used by a system like Gemini in the creation of new rules for a partial domain theory. This is obviously a subset of all possible heuristics that could be generated using the methodology described in Chapter 4. According to that methodology, if heuristics are viewed as rules with a condition for application and a specified action, there are theoretically 729 distinct conditions alone, each of which could result in a number of actions. Due to space limitations, only a subset was generated for this appendix. Here we describe the method followed in generating these heuristics, and explain the choices made in eliminating others.

Heuristics were generated by first building all possible conditions as described in Chapter 4. Once these were generated, a number were pruned due to the fact that they could never occur given the domain theories used in evaluating this dissertation. In our domain theories, if all rules used in two explanations are identical, their subgoals must be identical as well. That is, a set of rules uniquely specifies the set of subgoals concluded from them. All heuristic conditions requiring that two explanations have identical rules but different subgoals, for example, were removed as two such explanations could never be found (and thus the heuristic would never be used). Similarly, a single set of rules indicates a single unique goal concept.

Next, a set of possible conditions was pruned due to the fact that they occurred *extensively* in the diagnosis domains used in testing. As described in Chapter 4, it is quite common to find explanations of diagnoses which are identical except for the final inference to the goal (i.e., the diagnosis). This occurs in situations where examples are instances of a general problem class, but some single attribute allows for the final inference to more specific diagnoses. Because this condition appears often in diagnosis domains, a number of heuristics referring to it were selected for testing. Since they were discussed extensively in the text, we chose not to duplicate that general class of conditions here. That is, if a condition indicated shared subgoals, we eliminated all conditions that specified different or unknown goals.

Consider the following 6-tuple: (features, goal, subgoals, leaves, unproven subgoals, rules), representing the six attributes from which conditions on heuristics are derived. Relationships between examples are represented by indicating their similarity or difference with respect to these. According to the methodology of Chapter 4, a ‘‘don't-care’’ might be designated about any attribute as well. Ignore that for now, and consider the eight 6-tuples describing the possible relationships based upon the attributes of rules, subgoals, and goal as discussed in the preceding paragraphs:

```
(?, 0, 0, ?, ?, 0)    *
(?, 0, 0, ?, ?, 1)
(?, 0, 1, ?, ?, 0)
(?, 0, 1, ?, ?, 1)
(?, 1, 0, ?, ?, 0)    *
(?, 1, 0, ?, ?, 1)
(?, 1, 1, ?, ?, 0)    *
(?, 1, 1, ?, ?, 1)    *
```

In these, 1 indicates that two examples are the same in the attribute, while 0 indicates that they are different. Of these, only the four marked by asterisks were retained here. As discussed earlier, if all rules used in the explanations of two examples are the same, their subgoals and thus final goals must be the same as well. If all subgoals are the same, we consider only same goals in this appendix. Of the original conditions generated, only those with one of the above four relationships were retained, leaving $4(3^3) = 108$ of the original 729. As noted, we selected to disallow heuristics with a ‘‘don’t-care’’ in any of the three discussed attributes.

Next, another set of heuristic conditions was eliminated — again, due to the fact that those conditions would never be true in our research. This time the restriction was due only in part to our domain theories. It resulted largely from the manner in which heuristics are implemented in Gemini.

Consider two explanations in which all rules applied are the same. Next consider the leaves of those explanations. The leaves contain information about both the names of input features and their values. When determining whether the leaves of two explanations are the same or are different, Gemini looks only at the *names* of features mentioned. It does not consider values. In Gemini, if all rules are identical, they must refer to identical feature names. Thus the leaves to which those feature names correspond must be the same. Furthermore, if all rules used are the same and there exist one or more missing subgoals, those must be identical across the explanations as well. Consider the following eight 6-tuples describing the possible ‘‘same’’ and ‘‘different’’ relationships with respect to the leaves, unproven subgoals, and rules attributes just discussed:

```
(?, ?, ?, 0, 0, 0)   *
(?, ?, ?, 0, 0, 1)
(?, ?, ?, 0, 1, 0)   *
(?, ?, ?, 0, 1, 1)
(?, ?, ?, 1, 0, 0)   *
(?, ?, ?, 1, 0, 1)
(?, ?, ?, 1, 1, 0)   *
(?, ?, ?, 1, 1, 1)   *
```

Of these, only the five marked by asterisks might occur given our implementation of domain theories and heuristics. Non-occurring cases were eliminated from the set of conditions generated.

Recall that the previous phase left 108 of the original 729 conditions generated. This resulted from the fact that only four relationships were retained based upon the attributes of goals, subgoals, and rules. Consider the three of the four cases where the rules in the two examples under consideration are different. As shown above, any configuration of the remaining three attributes of features, leaves, and unproven subgoals may still occur. Including don't-cares, this accounts for $3(3^3)$ of the $4(3^3)$ = 108 possible conditions.

Now consider the one relationship of four where the rules in the two examples are the same. In this instance, there are restrictions on the relationships that might occur with respect to the remaining three attributes. Specifically, the unproven subgoal will never be different (leaving only similarity or don't-care as relevant relationships for a heuristic condition); similarly, different leaves will not be found. A relationship specified with respect to example features remains unrestricted. This leaves (3)(2)(2) of the 3^3 conditions generated earlier. The total number of heuristic conditions remaining after pruning is $3(3^3) + (3)(2)(2) = 93$.

For each of the 93 conditions a *single* action (or conclusion) was generated. An additional heuristic was added with no condition on application. (The non-restrictive condition had been eliminated earlier.)

Appendix B

Domain Specifications

This appendix gives detailed specifications of the four domains used in testing the hypotheses of this research: network fault diagnosis, radio fault diagnosis (military communications / manpack), radio fault diagnosis (military communications / vehicular), and terrorist event news stories. For each domain, the following are given: domain description, input frame description, sample input frames, and rule base. The domain details are preceded by a general description of Gemini's input frame syntax and rule syntax.

B.1. Input Frame Syntax

The following description of Gemini's input frame syntax is excerpted (and slightly modified) from the system documentation.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Gemini's input instances are described by a frame representation.
;; Each input consists of a frame with slots that may contain either
;; a single value or a pointer to another frame describing that slot
;; in more detail.  Thus each input may actually be a hierarchical
;; frame structure.
;;
;; Each input frame is an instance of a pre-declared frame type.  The
;; syntax for declaration of a frame type in Gemini is as follows:
;;
;;   (PUTPROP 'frame-type 'SLOTS
;;
;;      '((slot-name-1   slot-match-type-1)
;;        (slot-name-2   slot-match-type-2)
;;        (slot-name-3   slot-match-type-3)
;;        .
;;        .
;;        .
;;        (slot-name-n   slot-match-type-n)))
;;
;; where frame-type = the name of the frame type;
;;       slot-name  = the name to be given to a slot;
;;       slot-match-type = the method to be used if values in that
;;                    slot need to be generalized.  One might desire
;;                    to generalize symbolic values differently than
```

```
;;                     one would numeric or time values.  In the domains
;;                     described here, the generalization type
;;                     'generalize-slot is used almost exclusively.  It
;;                     indicates that a slot value is symbolic.
;;
;; To create an instance of a particular type of frame, the following
;; syntax is used:
;;
;;   (PUTPROP 'instance-name 'FRAME-TYPE 'frame-type-name)
;;     (FPUT 'instance-name 'slot-name-1 'VALUE 'slot-value)
;;     (FPUT 'instance-name 'slot-name-2 'VALUE 'slot-value)
;;     (FPUT 'instance-name 'slot-name-3 'VALUE 'slot-value)
;;     .
;;     .
;;     .
;;     (FPUT 'instance-name 'slot-name-n 'VALUE 'slot-value)
;;
;; where instance-name = the unique identifier to be given to the
;;          instance;
;;       frame-type-name = the name of a frame-type;
;;       slot-name = the name of a slot in the declared type;
;;       slot-value = the value of the slot for the declared instance.
;;          The slot-value may be either a single value or may be a
;;          pointer describing the slot in more detail.  If it is a
;;          single value, it is described using an atom.  If it is
;;          a pointer, it is described by a list of two atoms, where
;;          the first atom is the unique identifier for the subframe
;;          instance and the second atom is the type of the subframe
;;          instance.
;; A slot can be specified as having unknown value by simply
;; excluding any statement declaring its value.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

B.2. Rule Base Syntax

The following description of Gemini's rule syntax is excerpted from the system documentation.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; The rule base is a set of priority-ordered IF-THEN rules.  Rules
;; appearing earlier in the base will fire before those that appear
;; later.  Disjunctive conditions are represented as separate rules.
;; The format of rules is as follows:
;;
;;   (RULEnumber
;;     (IF
;;          (condition 1)
;;          (condition 2)
;;          .
;;          .
;;          .
;;          (condition n))
;;     (THEN
;;          (FACT subgoal-to-be-concluded))
;;     (CONTEXT ((path-of-nested-frame-types) slot context-name)
;;              .
```

```
;;              .
;;              .
;;              ((path-of-nested-frame-types) slot context-name)))
;;
;; A condition can be any one of the following:
;;
;;   (ISA     x? slot-name slot-value)        OR
;;   (NOT-ISA x? slot-name slot-value)        OR
;;   (HAS     x? slot-name slot-value)        OR
;;   (NOT-HAS x? slot-name slot-value)        OR
;;   (GEN     x? slot-name slot-value)        OR
;;   (NOT-GEN x? slot-name slot-value)        OR
;;   (EQUAL x? slot-name slot-value)          OR
;;   (NOT-EQUAL x? slot-name slot-value)      OR
;;   (LESS-THAN x? slot-name slot-value)      OR
;;   (NOT-LESS-THAN x? slot-name slot-value)  OR
;;   (GRTR-THAN x? slot-name slot-value)      OR
;;   (NOT-GRTR-THAN x? slot-name slot-value),
;;
;; where x? is a variable bound to an input frame, OR
;;
;;   (ISA x? (frame-path)),
;;
;; where x? is a variable to be bound to a frame of a specified
;; type.  The type is indicated by the frame path.  If
;; frame-path = (t1 t2 ... tn) then the variable is to be bound
;; to the frame pointed to by the slot tn where the input frame
;; has a slot t1 which points to a frame containing slot t2
;; ... which points to a frame containing slot tn which points to
;; a frame.
;;
;; Each condition specifies a relationship that must hold between
;; the input and a particular value.  The relationships may be
;; categorized into two groups, symbolic and numeric, as follows:
;;
;; SYMBOLIC CONDITIONS:
;;
;;   ISA - binds the variable to the top level input frame; then
;;     looks for exact match between slot-value and slot-name's value
;;     in the bound frame.
;;   NOT-ISA - binds the variable to the top level input frame;
;;     then checks slot-name's value in the bound frame to be sure
;;     that it is either not specified or not slot-value.
;;   HAS - looks for exact match between slot-value and slot-name's
;;     value in the bound frame.
;;   NOT-HAS - looks for non-existence of variable binding or
;;     non-existence or inequality of the specified slot value.
;;   GEN - checks that slot-name's value in the bound frame is a
;;     descendent of slot-value in the static semantic memory.
;;   NOT-GEN - tests for non-existence of the variable binding or
;;     that the specified slot value is not a specialization of
;;     the given match value.
;;
;; NUMERIC CONDITIONS:
;;
;;   EQUAL - tests that slot-name's value in the bound frame is a
;;     number equal to slot-value.
;;   NOT-EQUAL - tests for non-existence of variable binding or that
;;     the slot value is not a number equal to the given match value.
;;   LESS-THAN - tests that slot-name's value in the bound frame is
;;     a number less than the given match value.
;;   NOT-LESS-THAN - tests for non-existence of the variable binding
;;     or that the slot value is not a number less than the given
```

```
;;     match value.
;;   GRTR-THAN - tests that slot-name's value in the bound frame is
;;     a number greater than the given match value.
;;   NOT-GRTR-THAN - tests for non-existence of variable binding or
;;     that the slot value is not a number greater than the given
;;     match value.
;;
;; A condition may also be
;;
;;   (FACT subgoal-concluded),
;;
;; which indicates that a subgoal must be concluded before the rule
;; may apply.
;;
;; In the CONTEXT portion of a rule, the list containing
;; path-of-nested-frame-types is replaced by a single atom indicating
;; an appropriate frame type if frame binding is to occur with the
;; top-level input frame.  The context indicator is entirely optional.
;;
;; **SPECIAL NOTE** - All frame bindings remain in effect ONLY across
;; a single rule.  That is, they are used ONLY to determine whether
;; a single rule applies.  New bindings are established as each rule
;; is tested.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

B.3. Network Fault Diagnosis

In this domain, the given network is an ethernet/token ring network. A fault is signalled by the inability of one user to reach another connected to the same network. The knowledge used by the performance system is encoded on a level that enables isolation of a fault to a particular segment of a network, but does not allow deeper analysis of the problem.

A number of abbreviations are used in the implementation of the domain. The following is a translation of those abbreviations:

```
COND     condition
CONN     connect(ion)
DEST     destination
FBR      fiber optic
FM       from
INIT     initializ(e/ation)
MAU      multistation access unit
MSG      message
NET      network
PC       IBM PC
POSS     possible
PROB     problem
PWR      power
RPTR     repeater
```

```
RT        router
RTR       router
SRC       source
TGT       target
TRN       token ring (network)
USR       user
W/S       workstation
WRKSTN    workstation
```

Input Frame Representation for Network Fault Diagnosis

This is the declaration of the *fault* frame type used to describe input instances for the domain of network fault diagnosis. The atom *generalize-slot* indicates the type of generalization method to use on a given slot. It indicates that the slot value is a symbol rather than, for instance, a number.

```
(PUTPROP ’fault ’slots

   ’((complaint-user-cant-connect?    generalize-slot)
     (cond-is-it-a-single-remote-MAU? generalize-slot)
     (cond-rptrs-are-in-fault-domain? generalize-slot)
     (cond-segnet-usr-can-initialize? generalize-slot)
     (cond-TD-gets-PC-init-failure?   generalize-slot)
     (cond-tgt-node-responds-to-ping? generalize-slot)
     (td-all-nodes-gone-fault-domain? generalize-slot)
     (td-can-ping-nodes-but-not-user? generalize-slot)
     (td-can-ping-rtr-fm-Edinboro?    generalize-slot)
     (td-can-ping-rtr-from-MAU?       generalize-slot)
     (td-can-ping-source-net?         generalize-slot)
     (td-can-user-reach-another-net?  generalize-slot)
     (td-local-fbr-rptr-low-lite-on?  generalize-slot)
     (td-local-fbr-rptr-low-signal?   generalize-slot)
     (td-local-fbr-rptr-pwr-lite-off? generalize-slot)
     (td-local-loopback-test?         generalize-slot)
     (td-remot-fbr-rptr-pwr-lite-off? generalize-slot)
     (td-remote-fbr-rptr-low-lite-on? generalize-slot)
     (td-remote-loopback-test?        generalize-slot)
     (td-rt-console-hung/no-msg?      generalize-slot)
     (td-rtr-console-has-error-msg?   generalize-slot)
     (td-the-user-is-on-a-TRN?        generalize-slot)
     (td-the-user-is-on-an-Ethernet?  generalize-slot)
     (td-user-cant-communicate-at-all generalize-slot)
     (td-user-says-cant-initialize?   generalize-slot)
     (td-user-wall-connection-OK?     generalize-slot)
     (td-usr-gets-init-retry-failure? generalize-slot)))
```

This is an example of an input frame from the network fault diagnosis domain. It is an instance of a *poss-fiber-optic-cable-prob*.

```
(PUTPROP ’fault12 ’frame-type ’fault)
(FPUT ’fault12 ’complaint-user-cant-connect?    ’value ’t)
(FPUT ’fault12 ’cond-is-it-a-single-remote-MAU? ’value ’f)
(FPUT ’fault12 ’cond-rptrs-are-in-fault-domain? ’value ’f)
(FPUT ’fault12 ’cond-segnet-usr-can-initialize? ’value ’t)
```

```
(FPUT 'fault12 'cond-TD-gets-PC-init-failure?   'value 't)
(FPUT 'fault12 'cond-tgt-node-responds-to-ping? 'value 't)
(FPUT 'fault12 'td-all-nodes-gone-fault-domain? 'value 'f)
(FPUT 'fault12 'td-can-ping-nodes-but-not-user? 'value 'f)
(FPUT 'fault12 'td-can-ping-rtr-fm-Edinboro?    'value 't)
(FPUT 'fault12 'td-can-ping-rtr-from-MAU?       'value 't)
(FPUT 'fault12 'td-can-ping-source-net?         'value 't)
(FPUT 'fault12 'td-can-user-reach-another-net?  'value 'f)
(FPUT 'fault12 'td-local-fbr-rptr-low-lite-on?  'value 't)
(FPUT 'fault12 'td-local-fbr-rptr-low-signal?   'value 't)
(FPUT 'fault12 'td-local-fbr-rptr-pwr-lite-off? 'value 't)
(FPUT 'fault12 'td-local-loopback-test?         'value 't)
(FPUT 'fault12 'td-remot-fbr-rptr-pwr-lite-off? 'value 'f)
(FPUT 'fault12 'td-remote-fbr-rptr-low-lite-on? 'value 'f)
(FPUT 'fault12 'td-remote-loopback-test?        'value 'f)
(FPUT 'fault12 'td-rt-console-hung/no-msg?      'value 'f)
(FPUT 'fault12 'td-rtr-console-has-error-msg?   'value 'f)
(FPUT 'fault12 'td-the-user-is-on-a-TRN?        'value 't)
(FPUT 'fault12 'td-the-user-is-on-an-Ethernet?  'value 'f)
(FPUT 'fault12 'td-user-cant-communicate-at-all 'value 'f)
(FPUT 'fault12 'td-user-says-cant-initialize?   'value 't)
(FPUT 'fault12 'td-user-wall-connection-OK?     'value 't)
(FPUT 'fault12 'td-usr-gets-init-retry-failure? 'value 'f)
```

Rule Base for Network Fault Diagnosis

The following is a listing of the rule base for the network fault diagnosis domain. We translated rules into Gemini's representation from a prototype knowledge base that was extracted from experts maintaining the CMU campus computer network [Eshelman 88].

```
(RULE1
   (IF
        (isa x? complaint-user-cant-connect? t)
        (isa x? cond-tgt-node-responds-to-ping? t)
        (isa x? td-can-user-reach-another-net? f))
   (THEN
        (fact looks-like-source-prob)))

(RULE2
   (IF
        (isa x? complaint-user-cant-connect? t)
        (isa x? td-can-user-reach-another-net? t))
   (THEN
        (fact looks-like-dest-prob)))

(RULE3
   (IF
        (isa x? td-the-user-is-on-a-TRN? t)
        (fact looks-like-source-prob))
   (THEN
        (fact looks-like-source-prob-TRN)))

(RULE4
   (IF
        (isa x? td-the-user-is-on-an-Ethernet? t)
        (fact looks-like-source-prob))
   (THEN
        (fact looks-like-source-prob-Ethernet)))
```

```
(RULE5
   (IF
        (isa x? td-can-ping-source-net? t))
   (THEN
        (fact not-poss-isolated-net-SRC-TRN)))

(RULE6
   (IF
        (isa x? td-can-ping-source-net? t)
        (fact looks-like-source-prob-TRN))
   (THEN
        (fact poss-segmented-SRC-TRN)))

(RULE7
   (IF
        (isa x? td-can-ping-nodes-but-not-user? t)
        (fact looks-like-source-prob-TRN))
   (THEN
        (fact poss-segmented-SRC-TRN)))

(RULE8
   (IF
        (isa x? td-can-ping-source-net? f)
        (fact looks-like-source-prob-TRN))
   (THEN
        (fact poss-isolated-net-SRC-TRN)))

(RULE9
   (IF
        (isa x? cond-TD-gets-PC-init-failure? t)
        (isa x? td-user-says-cant-initialize? t)
        (fact looks-like-source-prob-TRN))
   (THEN
        (fact poss-failure-SRC-TRN)))

(RULE10
   (IF
        (isa x? td-can-ping-source-net? f)
        (fact looks-like-source-prob-TRN))
   (THEN
        (fact poss-failure-SRC-TRN)))

(RULE11
   (IF
        (isa x? td-usr-gets-init-retry-failure? t)
        (fact looks-like-source-prob-TRN))
   (THEN
        (fact poss-non-operative-node-SRC-TRN)))

(RULE12
   (IF
        (isa x? td-user-says-cant-initialize? t)
        (fact looks-like-source-prob-TRN))
   (THEN
        (fact poss-non-operative-node-SRC-TRN)))

(RULE13
   (IF
        (isa x? td-can-ping-source-net? f))
   (THEN
        (fact not-poss-segmented-SRC-TRN)))

(RULE14
```

```
   (IF
        (isa x? cond-segnet-usr-can-initialize? t)
        (isa x? td-user-cant-communicate-at-all t)
        (fact poss-segmented-SRC-TRN))
   (THEN
        (fact poss-segmented-net-1user)))

(RULE15
   (IF
        (fact poss-segmented-SRC-TRN))
   (THEN
        (fact poss-segmented-net-multi-users)))

(RULE16
   (IF
        (isa x? td-rtr-console-has-error-msg? t)
        (fact poss-isolated-net-SRC-TRN))
   (THEN
        (fact poss-backbone->SRC-router-prob)))

(RULE17
   (IF
        (fact poss-failure-SRC-TRN))
   (THEN
        (fact poss-local-failure)))

(RULE18
   (IF
        (fact poss-failure-SRC-TRN))
   (THEN
        (fact poss-remote-failure)))

(RULE19
   (IF
        (isa x? td-local-fbr-rptr-low-lite-on? t)
        (fact poss-failure-SRC-TRN))
   (THEN
        (fact poss-fiber-optic-cable-prob)))

(RULE20
   (IF
        (isa x? td-remote-fbr-rptr-low-lite-on? t)
        (fact poss-failure-SRC-TRN))
   (THEN
        (fact poss-fiber-optic-cable-prob)))

(RULE21
   (IF
        (fact poss-non-operative-node-SRC-TRN))
   (THEN
        (fact poss-user-w/s-software-prob)))

(RULE22
   (IF
        (fact poss-non-operative-node-SRC-TRN))
   (THEN
        (fact poss-user-w/s-hardware-prob)))

(RULE23
   (IF
        (isa x? td-rt-console-hung/no-msg? t)
        (fact poss-isolated-net-SRC-TRN))
   (THEN
```

```
        (fact poss-SRC-router-prob)))

(RULE24
   (IF
        (isa x? td-can-ping-rtr-from-Edinboro? f)
        (fact poss-isolated-net-SRC-TRN))
   (THEN
        (fact poss-SRC-router-prob)))

(RULE25
   (IF
        (isa x? td-can-ping-rtr-from-MAU? f)
        (fact poss-isolated-net-SRC-TRN))
   (THEN
        (fact poss-SRC-router-prob)))

(RULE26
   (IF
        (isa x? td-local-fbr-rptr-low-signal? t)
        (fact poss-isolated-net-SRC-TRN))
   (THEN
        (fact poss-SRC-router-prob)))

(RULE27
   (IF
        (isa x? td-can-ping-rtr-from-Edinboro? f)
        (fact poss-local-failure))
   (THEN
        (fact poss-SRC-router-prob)))

(RULE28
   (IF
        (isa x? td-rt-console-hung/no-msg? t)
        (fact poss-local-failure))
   (THEN
        (fact poss-SRC-router-prob)))

(RULE29
   (IF
        (isa x? td-can-ping-rtr-from-MAU? f)
        (fact poss-local-failure))
   (THEN
        (fact poss-SRC-router-prob)))

(RULE30
   (IF
        (isa x? td-local-fbr-rptr-low-signal? t)
        (fact poss-local-failure))
   (THEN
        (fact poss-SRC-router-prob)))

(RULE31
   (IF
        (fact poss-segmented-net-multi-users))
   (THEN
        (fact poss-cabling-prob-wire-closet)))

(RULE32
   (IF
        (fact poss-segmented-net-multi-users))
   (THEN
        (fact poss-cabling-prob-user-office)))
```

```
(RULE33
   (IF
        (isa x? td-can-ping-rtr-from-MAU? f)
        (fact poss-isolated-net-SRC-TRN))
   (THEN
        (fact poss-SRC-router->TRN-prob)))

(RULE34
   (IF
        (isa x? td-rtr-console-has-error-msg? t)
        (fact poss-isolated-net-SRC-TRN))
   (THEN
        (fact poss-SRC-router->TRN-prob)))

(RULE35
   (IF
        (isa x? td-local-fbr-rptr-low-signal? t)
        (fact poss-isolated-net-SRC-TRN))
   (THEN
        (fact poss-SRC-router->TRN-prob)))

(RULE36
   (IF
        (isa x? td-can-ping-rtr-from-MAU? f)
        (fact poss-local-failure))
   (THEN
        (fact poss-SRC-router->TRN-prob)))

(RULE37
   (IF
        (isa x? td-rtr-console-has-error-msg? t)
        (fact poss-local-failure))
   (THEN
        (fact poss-SRC-router->TRN-prob)))

(RULE38
   (IF
        (isa x? td-local-fbr-rptr-low-signal? t)
        (fact poss-local-failure))
   (THEN
        (fact poss-SRC-router->TRN-prob)))

(RULE39
   (IF
        (isa x? td-user-wall-connection-OK? t)
        (fact poss-failure-SRC-TRN))
   (THEN
        (fact poss-user-wrkstn->net-conn-prob)))

(RULE40
   (IF
        (isa x? td-user-wall-connection-OK? t)
        (fact poss-segmented-net-1user))
   (THEN
        (fact poss-user-wrkstn->net-conn-prob)))

(RULE41
   (IF
        (isa x? td-user-wall-connection-OK? t)
        (fact poss-non-operative-node-SRC-TRN))
   (THEN
        (fact poss-user-wrkstn->net-conn-prob)))
```

```
(RULE42
   (IF
        (isa x? td-local-fbr-rptr-low-signal? t)
        (fact poss-local-failure))
   (THEN
        (fact poss-local-MAU-prob)))

(RULE43
   (IF
        (isa x? td-local-loopback-test? t)
        (fact poss-local-failure))
   (THEN
        (fact poss-local-fiber-rptr-prob)))

(RULE44
   (IF
        (isa x? td-local-fbr-rptr-low-signal? t)
        (fact poss-local-failure))
   (THEN
        (fact poss-local-fiber-rptr-prob)))

(RULE45
   (IF
        (isa x? td-local-fbr-rptr-pwr-lite-off? t)
        (fact poss-local-failure))
   (THEN
        (fact poss-local-fiber-rptr-prob)))

(RULE46
   (IF
        (isa x? td-local-fbr-rptr-low-lite-on? t)
        (fact poss-local-failure))
   (THEN
        (fact poss-local-fiber-rptr-prob)))

(RULE47
   (IF
        (isa x? td-remote-fbr-rptr-low-lite-on? t)
        (fact poss-local-failure))
   (THEN
        (fact poss-local-fiber-rptr-prob)))

(RULE48
   (IF
        (isa x? td-remote-loopback-test? t)
        (fact poss-remote-failure))
   (THEN
        (fact poss-remote-fiber-rptr-prob)))

(RULE49
   (IF
        (isa x? td-local-fbr-rptr-low-lite-on? t)
        (fact poss-remote-failure))
   (THEN
        (fact poss-remote-fiber-rptr-prob)))

(RULE50
   (IF
        (isa x? td-remote-fbr-rptr-low-lite-on? t)
        (fact poss-remote-failure))
   (THEN
        (fact poss-remote-fiber-rptr-prob)))
```

```
(RULE51
   (IF
        (isa x? td-remot-fbr-rptr-pwr-lite-off? t)
        (fact poss-remote-failure))
   (THEN
        (fact poss-remote-fiber-rptr-prob)))

(RULE52
   (IF
        (fact poss-remote-failure))
   (THEN
        (fact poss-remote-TRN-prob)))

(RULE53
   (IF
        (isa x? cond-rptrs-are-in-fault-domain? t)
        (fact poss-remote-TRN-prob))
   (THEN
        (fact poss-remote-copper-rptr-prob)))

(RULE54
   (IF
        (fact poss-remote-TRN-prob))
   (THEN
        (fact poss-remote-interface-failure)))

(RULE55
   (IF
        (fact poss-remote-TRN-prob))
   (THEN
        (fact poss-remote-cabling-failure)))

(RULE56
   (IF
        (isa x? cond-is-it-a-single-remote-MAU? t)
        (isa x? td-all-nodes-gone-fault-domain? t)
        (fact poss-remote-TRN-prob))
   (THEN
        (fact poss-remote-MAU-prob)))
```

B.4. Radio Fault Diagnosis: Military Communication / Manpack

In this domain, the radio diagnosed is a communication radio used by the military. It is a pack radio, to be carried by an individual. As in the network fault domain, knowledge is encoded on a level that enables isolation of the fault to a particular major radio component, but does not allow deeper analysis of the problem. The rule base for this domain is our direct translation of troubleshooting procedures listed in the manual for the radio [Radio Manual 86].

A number of abbreviations are used in the implementation of the domain. The following is a translation of those abbreviations:

```
A         Action
COMSEC    COMmunication SECurity
D         Diagnosis
ECCM      Electronic Counter CounterMeasures
P         Partial diagnosis
PTT       Push To Talk
R         test Result
RT        Receiver-Transmitter
S         Symptom
```

Input Frame Representation for Radio Fault Diagnosis - Manpack

This is the declaration of the *fault* frame type used to describe input instances for the domain of manpack radio fault diagnosis:

```
(PUTPROP 'fault 'slots

   ;; SYMPTOMS
  '((S-loss-of-memory                                  generalize-slot)
    (S-loss-of-sidetone                                generalize-slot)
    (S-loss-of-receive-audio                           generalize-slot)
    (S-eccm-data-will-not-load                         generalize-slot)
    (S-rt-not-operating-with-digital-data-devices      generalize-slot)
    (S-KY57-alarm-tone-or-beep-not-heard-at-rt         generalize-slot)
    (S-rt-loses-memory-when-rt-off                     generalize-slot)

    ;; RESULTS OF TESTS
    (R-response-with-new-holding-battery               generalize-slot)
    (R-response-with-new-antenna                       generalize-slot)
    (R-response-with-old-antenna-base                  generalize-slot)
    (R-response-with-new-handset                       generalize-slot)
    (R-response-with-new-eccm-fill-device              generalize-slot)
    (R-failed-rt-self-test                             generalize-slot)
    (R-failed-digital-device-self-test                 generalize-slot)
    (R-interface-cable-observed-opens-shorts           generalize-slot)
    (R-response-with-new-digital-device                generalize-slot)
    (R-response-with-reconnect-KY57                    generalize-slot)
    (R-response-to-push-handset-ptt-switch             generalize-slot)
    (R-comsec-cable-observed-opens-shorts              generalize-slot)
    (R-response-with-new-rt                            generalize-slot)
    (R-response-with-new-holding-battery-long-off-time generalize-slot)
     ))
```

This is an example of an input frame from the manpack radio fault diagnosis domain. It is an instance of *d-holding-battery-bad*, i.e., a bad holding battery. Note that a number of the possible features have been left unspecified. The value of a feature may be unknown. That can be represented either by explicitly assigning a feature the value *nil*, or by not declaring any value for a feature.

```
(PUTPROP 'fault2 'frame-type 'fault)
(FPUT 'fault2 'S-loss-of-memory                    'value 't)
(FPUT 'fault2 'S-rt-loses-memory-when-rt-off       'value 't)

(FPUT 'fault2 'R-response-with-new-holding-battery   'value 'YES)
```

```
(FPUT 'fault2 'R-response-with-new-handset             'value 'NO)
(FPUT 'fault2 'R-failed-rt-self-test                   'value 'NO)
(FPUT 'fault2 'R-interface-cable-observed-opens-shorts 'value 'NO)
```

Rule Base for Radio Fault Diagnosis - Manpack

The following is a listing of the rule base for the domain of manpack radio fault diagnosis.

```
(RULE1
   (IF
        (isa x? S-loss-of-memory t))
   (THEN
        (fact P-rt-or-holding-battery-bad
              A-try-new-holding-battery)))

(RULE2
   (IF
        (fact P-rt-or-holding-battery-bad)
        (isa x? R-response-with-new-holding-battery NO))
   (THEN
        (fact D-rt-bad)))

(RULE3
   (IF
        (fact P-rt-or-holding-battery-bad)
        (isa x? R-response-with-new-holding-battery YES))
   (THEN
        (fact D-holding-battery-bad)))

(RULE4
   (IF
        (isa x? S-loss-of-sidetone t))
   (THEN
        (fact P-rt-or-antenna-bad
              A-try-new-antenna)))

(RULE5
   (IF
        (fact P-rt-or-antenna-bad)
        (isa x? R-response-with-new-antenna NO))
   (THEN
        (fact D-rt-bad)))

(RULE6
   (IF
        (fact P-rt-or-antenna-bad)
        (isa x? R-response-with-new-antenna YES))
   (THEN
        (fact P-antenna-base-or-antenna-element-bad
              A-try-old-antenna-base)))

(RULE7
   (IF
        (fact P-antenna-base-or-antenna-element-bad)
        (isa x? R-response-with-old-antenna-base NO))
   (THEN
        (fact D-antenna-base-bad)))

(RULE8
   (IF
```

```
        (fact P-antenna-base-or-antenna-element-bad)
        (isa x? R-response-with-old-antenna-base YES))
   (THEN
        (fact D-antenna-element-bad)))

(RULE9
   (IF
        (isa x? S-loss-of-receive-audio t))
   (THEN
        (fact P-rt-or-handset-bad
              A-try-new-handset)))

(RULE10
   (IF
        (fact P-rt-or-handset-bad)
        (isa x? R-response-with-new-handset NO))
   (THEN
        (fact D-rt-bad)))

(RULE11
   (IF
        (fact P-rt-or-handset-bad)
        (isa x? R-response-with-new-handset YES))
   (THEN
        (fact D-handset-bad)))

(RULE12
   (IF
        (isa x? S-eccm-data-will-not-load t))
   (THEN
        (fact P-rt-or-eccm-fill-device-bad
              A-try-new-eccm-fill-device)))

(RULE13
   (IF
        (fact P-rt-or-eccm-fill-device-bad)
        (isa x? R-response-with-new-eccm-fill-device NO))
   (THEN
        (fact D-rt-bad)))

(RULE14
   (IF
        (fact P-rt-or-eccm-fill-device-bad)
        (isa x? R-response-with-new-eccm-fill-device YES))
   (THEN
        (fact D-eccm-fill-device-bad)))

(RULE15
   (IF
        (isa x? S-rt-not-operating-with-digital-data-devices t))
   (THEN
        (fact P-rt-or-digital-device-or-cable-bad-prefer-rt
              A-try-rt-self-test)))

(RULE16
   (IF
        (fact P-rt-or-digital-device-or-cable-bad-prefer-rt)
        (isa x? R-failed-rt-self-test YES))
   (THEN
        (fact D-rt-bad)))

(RULE17
   (IF
```

```
        (fact P-rt-or-digital-device-or-cable-bad-prefer-rt)
        (isa x? R-failed-rt-self-test NO))
   (THEN
        (fact P-rt-or-digital-device-or-cable-bad-prefer-device
              A-try-digital-device-self-test)))

(RULE18
   (IF
        (fact P-rt-or-digital-device-or-cable-bad-prefer-device)
        (isa x? R-failed-digital-device-self-test YES))
   (THEN
        (fact D-digital-device-bad)))

(RULE19
   (IF
        (fact P-rt-or-digital-device-or-cable-bad-prefer-device)
        (isa x? R-failed-digital-device-self-test NO))
   (THEN
        (fact P-rt-or-digital-device-or-cable-bad-prefer-cable
              A-look-at-interface-cable-for-opens-and-shorts)))

(RULE20
   (IF
        (fact P-rt-or-digital-device-or-cable-bad-prefer-cable)
        (isa x? R-interface-cable-observed-opens-shorts YES))
   (THEN
        (fact D-interface-cable-bad)))

(RULE21
   (IF
        (fact P-rt-or-digital-device-or-cable-bad-prefer-cable)
        (isa x? R-interface-cable-observed-opens-shorts NO))
   (THEN
        (fact P-rt-or-digital-device-bad
              A-try-new-digital-device)))

(RULE22
   (IF
        (fact P-rt-or-digital-device-bad)
        (isa x? R-response-with-new-digital-device NO))
   (THEN
        (fact D-rt-bad)))

(RULE23
   (IF
        (fact P-rt-or-digital-device-bad)
        (isa x? R-response-with-new-digital-device YES))
   (THEN
        (fact D-digital-device-bad)))

(RULE24
   (IF
        (isa x? S-KY57-alarm-tone-or-beep-not-heard-at-rt t))
   (THEN
        (fact P-rt-or-KY57-or-comsec-bad-prefer-KY57
              A-try-reconnect-KY57)))

(RULE25
   (IF
        (fact P-rt-or-KY57-or-comsec-bad-prefer-KY57)
        (isa x? R-response-with-reconnect-KY57 NO))
   (THEN
        (fact D-KY57-bad)))
```

```
(RULE26
   (IF
        (fact P-rt-or-KY57-or-comsec-bad-prefer-KY57)
        (isa x? R-response-with-reconnect-KY57 YES))
   (THEN
        (fact P-rt-or-KY57-or-comsec-bad-try-handset
              A-push-handset-ptt-switch)))

(RULE27
   (IF
        (fact P-rt-or-KY57-or-comsec-bad-try-handset)
        (isa x? R-response-to-push-handset-ptt-switch YES))
   (THEN
        (fact D-KY57-bad)))

(RULE28
   (IF
        (fact P-rt-or-KY57-or-comsec-bad-try-handset)
        (isa x? R-response-to-push-handset-ptt-switch NO))
   (THEN
        (fact P-rt-or-KY57-or-comsec-bad-prefer-cable
              A-look-at-comsec-cable-for-opens-and-shorts)))

(RULE29
   (IF
        (fact P-rt-or-KY57-or-comsec-bad-prefer-cable)
        (isa x? R-comsec-cable-observed-opens-shorts YES))
   (THEN
        (fact D-comsec-cable-bad)))

(RULE30
   (IF
        (fact P-rt-or-KY57-or-comsec-bad-prefer-cable)
        (isa x? R-comsec-cable-observed-opens-shorts NO))
   (THEN
        (fact P-rt-or-KY57-bad
              A-try-new-rt)))

(RULE31
   (IF
        (fact P-rt-or-KY57-bad)
        (isa x? R-response-with-new-rt NO))
   (THEN
        (fact D-KY57-bad)))

(RULE32
   (IF
        (fact P-rt-or-KY57-bad)
        (isa x? R-response-with-new-rt YES))
   (THEN
        (fact D-rt-bad)))

(RULE33
   (IF
        (isa x? S-rt-loses-memory-when-rt-off t))
   (THEN
        (fact P-rt-or-holding-battery-bad
              A-try-new-holding-battery-long-off-time)))

(RULE34
   (IF
        (fact P-rt-or-holding-battery-bad)
        (isa x? R-response-with-new-holding-battery-long-off-time YES))
```

```
   (THEN
        (fact D-holding-battery-bad)))

(RULE35
   (IF
        (fact P-rt-or-holding-battery-bad)
        (isa x? R-response-with-new-holding-battery-long-off-time NO))
   (THEN
        (fact D-rt-bad)))
```

B.5. Radio Fault Diagnosis: Military Communication / Vehicular

In this domain, the radio diagnosed is a communication radio used by the military. There are seven configurations of radio sets built around a single specific receiver-transmitter. One is the manpack radio, described above. The remaining six configurations are vehicular radios. Any of the six can be represented using the input frame developed for this domain. Note that the input representation is a rough model of the radio rather than a listing of symptoms used for the manpack radio. The representation does not, however, model all components of vehicular radios.

A number of abbreviations are used in the implementation of the domain. The following is a translation of those abbreviations:

```
cb1            on/off switch
ds1            dimmable light
fctn           function (switch)
neg            negative
pos            positive
poss           possible
prob           problem
rf             radio frequency
rt             receiver-transmitter
term           terminal
X-YZ-resist    resistance from pin Y to pin Z
               on connector x
```

Input Frame Representation for Radio Fault Diagnosis - Vehicular

This is the declaration of the *fault* frame type used to describe input instances for the domain of vehicular radio fault diagnosis:

```
(PUTPROP 'fault 'slots '((radioA          generalize-slot)
                         (radioB          generalize-slot)
                         (vehicle-battery generalize-slot)))
```

In this domain, inputs are represented by hierarchical frames. That is, features in this domain point to additional frames that describe them in more detail. The following are declarations of various frame types that may be used to describe components of the composite radio in more detail:

```
(PUTPROP 'radioA 'slots  '((type             generalize-slot)
                           (rtA              generalize-slot)
                           (mounting-adapter generalize-slot)
                           (comsec-mount     generalize-slot)
                           (mounting-base    generalize-slot)
                           (pa-mount         generalize-slot)
                           (connection       generalize-slot)))

(PUTPROP 'radioB 'slots  '((type             generalize-slot)
                           (rtA              generalize-slot)
                           (mounting-adapter generalize-slot)
                           (comsec-mount     generalize-slot)
                           (mounting-base    generalize-slot)
                           (pa-mount         generalize-slot)
                           (connection       generalize-slot)))

(PUTPROP 'vehicle-battery 'slots '((pos-term number)   ;measured
                                                       ;voltage
                                   (neg-term number)))

(PUTPROP 'rtA 'slots '((display        generalize-slot)
                       (fctn           generalize-slot)
                       (mode           generalize-slot)
                       (rf-power-level generalize-slot)
                       (volume         generalize-slot)))

(PUTPROP 'mounting-adapter 'slots '((cb1          generalize-slot)
                                    (ds1          generalize-slot)
                                    (p1-AB-resist number)))

(PUTPROP 'comsec-mount 'slots '((cb1 generalize-slot)
                                (ds1 generalize-slot)))

(PUTPROP 'mounting-base 'slots '((j1-AB-resist number)
                                 (j5-BA-resist number)))

(PUTPROP 'pa-mount 'slots '((cb1          generalize-slot)
                            (ds1          generalize-slot)
                            (j2-BA-resist number)))
```

This is an example of an input frame from the vehicular radio fault diagnosis domain. It is an instance of *ds1-lamp-faulty*, i.e., a bad dimmable light. Note that a number of the possible features have been left unspecified.

```
(PUTPROP 'fault1 'frame-type 'fault)
(FPUT 'fault1 'radioA          'value '(radioA1 radioA))
(FPUT 'fault1 'radioB          'value '(radioB1 radioB))
(FPUT 'fault1 'vehicle-battery 'value '(vehicle-battery1
                                        vehicle-battery))

(PUTPROP 'radioA1 'frame-type 'radioA)
```

```
(FPUT 'radioA1 'type             'value nil)
(FPUT 'radioA1 'rtA              'value '(rtAA1 rtA))
(FPUT 'radioA1 'mounting-adapter 'value '(mounting-adapterA1
                                          mounting-adapter))
(FPUT 'radioA1 'comsec-mount     'value '(comsec-mountA1
                                          comsec-mount))
(FPUT 'radioA1 'mounting-base    'value '(mounting-baseA1
                                          mounting-base))
(FPUT 'radioA1 'pa-mount         'value '(pa-mountA1 pa-mount))
(FPUT 'radioA1 'connection       'value nil)

(PUTPROP 'rtAA1 'frame-type 'rtA)
(FPUT 'rtAA1 'display        'value 'none)
(FPUT 'rtAA1 'fctn           'value 'SQ-ON)
(FPUT 'rtAA1 'mode           'value 'SC)
(FPUT 'rtAA1 'rf-power-level 'value 'LO)
(FPUT 'rtAA1 'volume         'value 'mid-range)

(PUTPROP 'mounting-adapterA1 'frame-type 'mounting-adapter)
(FPUT 'mounting-adapterA1 'cb1          'value 'on)
(FPUT 'mounting-adapterA1 'ds1          'value 'unlit)
(FPUT 'mounting-adapterA1 'p1-AB-resist 'value 1.0)

(PUTPROP 'comsec-mountA1 'frame-type 'comsec-mount)
(FPUT 'comsec-mountA1 'cb1 'value 'on)
(FPUT 'comsec-mountA1 'ds1 'value 'lit)

(PUTPROP 'mounting-baseA1 'frame-type 'mounting-base)
(FPUT 'mounting-baseA1 'j1-AB-resist 'value 1.0)
(FPUT 'mounting-baseA1 'j5-BA-resist 'value 1.0)

(PUTPROP 'pa-mountA1 'frame-type 'pa-mount)
(FPUT 'pa-mountA1 'cb1          'value 'on)
(FPUT 'pa-mountA1 'ds1          'value 'lit)
(FPUT 'pa-mountA1 'j2-BA-resist 'value 1.0)

(PUTPROP 'radioB1 'frame-type 'radioB)
(FPUT 'radioB1 'type             'value nil)
(FPUT 'radioB1 'rtA              'value nil)
(FPUT 'radioB1 'mounting-adapter 'value nil)
(FPUT 'radioB1 'comsec-mount     'value nil)
(FPUT 'radioB1 'mounting-base    'value nil)
(FPUT 'radioB1 'pa-mount         'value nil)
(FPUT 'radioB1 'connection       'value 'radioA)

(PUTPROP 'vehicle-battery1 'frame-type 'vehicle-battery)
(FPUT 'vehicle-battery1 'pos-term 'value 23)
(FPUT 'vehicle-battery1 'neg-term 'value 23)
```

Rule Base for Radio Fault Diagnosis - Vehicular

The following is a listing of the rule base for the domain of vehicular radio fault diagnosis. We translated these rules from troubleshooting charts in the radio manual [Radio Manual 86]. The rule base does not cover all troubleshooting procedures, containing only diagnostic rules for one class of possible faults.

```
(RULE1
   (IF
        (isa x? (radioA mounting-adapter))
```

```
        (has x? cb1 on)
        (has x? ds1 unlit))
   (THEN
        (fact poss-ds1-lamp-prob)))

(RULE2
   (IF
        (fact poss-ds1-lamp-prob)
        (isa x? (radioA rtA))
        (has x? fctn sq-on)
        (has x? display none))
   (THEN
        (fact ds1-lamp-faulty)))

(RULE3
   (IF
        (fact poss-ds1-lamp-prob)
        (isa x? (radioA rtA))
        (has x? fctn sq-on)
        (not-has x? display none)
        (not-has x? display nil))
   (THEN
        (fact poss-vehicle-battery-prob)))

(RULE4
   (IF
        (fact poss-vehicle-battery-prob)
        (isa x? (vehicle-battery))
        (less-than x? pos-term 22))
   (THEN
        (fact vehicle-battery-faulty)))

(RULE5
   (IF
        (fact poss-vehicle-battery-prob)
        (isa x? (vehicle-battery))
        (grtr-than x? pos-term 32))
   (THEN
        (fact vehicle-battery-faulty)))

(RULE6
   (IF
        (fact poss-vehicle-battery-prob)
        (isa x? (vehicle-battery))
        (less-than x? neg-term 22))
   (THEN
        (fact vehicle-battery-faulty)))

(RULE7
   (IF
        (fact poss-vehicle-battery-prob)
        (isa x? (vehicle-battery))
        (grtr-than x? neg-term 32))
   (THEN
        (fact vehicle-battery-faulty)))

(RULE8
   (IF
        (fact poss-vehicle-battery-prob)
        (isa x? (vehicle-battery))
        (not-less-than x? pos-term 22)
        (not-grtr-than x? pos-term 32)
        (not-less-than x? neg-term 22)
```

```
        (not-grtr-than x? neg-term 32))
   (THEN
        (fact poss-comsec-mount-prob)))

(RULE9
   (IF
        (fact poss-comsec-mount-prob)
        (isa x? (radioA comsec-mount))
        (has x? cb1 on)
        (has x? ds1 unlit))
   (THEN
        (fact poss-mounting-base-or-adapter-prob)))

(RULE10
   (IF
        (fact poss-mounting-base-or-adapter-prob)
        (isa x? (radioA mounting-base))
        (not-equal x? j1-AB-resist 1.0))
   (THEN
        (fact poss-mounting-base-prob)))

(RULE11
   (IF
        (fact poss-mounting-base-or-adapter-prob)
        (isa x? (radioA mounting-base))
        (equal x? j1-AB-resist 1.0))
   (THEN
        (fact poss-mounting-adapter-prob)))

(RULE12
   (IF
        (fact poss-mounting-base-prob)
        (isa x? (radioA mounting-base))
        (equal x? j5-BA-resist 1.0))
   (THEN
        (fact poss-radioB-prob)))

(RULE13
   (IF
        (fact poss-mounting-base-prob)
        (isa x? (radioA mounting-base))
        (not-equal x? j5-BA-resist 1.0))
   (THEN
        (fact mounting-base-faulty)))

(RULE14
   (IF
        (fact poss-mounting-adapter-prob)
        (isa x? (radioA mounting-adapter))
        (not-equal x? p1-AB-resist 1.0))
   (THEN
        (fact mounting-adapter-faulty)))

(RULE15
   (IF
        (fact poss-mounting-adapter-prob)
        (isa x? (radioA mounting-adapter))
        (equal x? p1-AB-resist 1.0))
   (THEN
        (fact power-cable-at-mounting-adapter-j1-faulty)))

(RULE16
   (IF
```

```
        (fact poss-radioB-prob)
        (isa x? (radioB))
        (has x? connection radioA))
   (THEN
        (fact poss-pa-mount-prob)))

(RULE17
   (IF
        (fact poss-radioB-prob)
        (isa x? (radioB))
        (not-has x? connection radioA)
        (not-has x? connection nil))
   (THEN
        (fact power-cable-at-mounting-base-A-j2-faulty)))

(RULE18
   (IF
        (fact poss-pa-mount-prob)
        (isa x? (radioA pa-mount))
        (not-equal x? j2-BA-resist 1.0))
   (THEN
        (fact pa-mount-faulty)))
```

B.6. Terrorist Event News Stories

This domain is modeled loosely around the following performance task: Given information from a news story describing a terrorist event, suggest an action that a law enforcement agency might take in response, given a database of earlier events. Input frames in this domain are translated by hand from newspaper stories into Gemini's representation. Gemini's goal is to explain why each event is a terrorist event, as opposed to some other type of news event. We created the rule base as an early test set for Gemini.

This domain differs from the preceding three domains in a number of ways: Though it is a classification domain, it is not a diagnosis domain. It is more complex in that inputs are hierarchical frames. Values of attributes for this domain require that relationships among them be defined in a static semantic memory.

Input Frame Representation for Terrorist Event News Stories

This is the declaration of the *event* frame type used to describe the highest level of input instances for the domain of terrorist event news stories.

```
(PUTPROP 'event 'slots '((type     generalize-slot)
                         (cause    generalize-slot)
                         (location generalize-slot)
                         (time     generalize-slot)
                         (target   generalize-slot)
                         (result   generalize-slot)
```

```
                         (claims   generalize-slot)
                         (warning  generalize-slot)))
```

The following are declarations of various frame types that may be used to describe high level features of the input events in more detail.

```
(PUTPROP 'cause 'slots '((instrument generalize-slot)
                         (rev-action generalize-slot)))

(PUTPROP 'location 'slots '((country generalize-slot)
                            (city    generalize-slot)
                            (place   generalize-slot)))

(PUTPROP 'time 'slots '((year       time)
                        (month      generalize-slot)
                        (day        time)
                        (hour       time)
                        (difference time)))

(PUTPROP 'target 'slots '((target_name generalize-slot)))

(PUTPROP 'result 'slots '((no_killed           number)
                          (killed_list         generalize-slot)
                          (no_hurt_seriously   number)
                          (hurt_seriously_list generalize-slot)
                          (no_hurt             number)
                          (hurt_list           generalize-slot)
                          (no_hostages         number)
                          (hostages_list       generalize-slot)
                          (damage              generalize-slot)))

(PUTPROP 'claims 'slots '((claimer generalize-slot)
                          (type    generalize-slot)
                          (method  generalize-slot)
                          (motives generalize-slot)
                          (demand  generalize-slot)))

(PUTPROP 'warning 'slots '((warner generalize-slot)
                           (method generalize-slot)))

(PUTPROP 'place 'slots '((environ       generalize-slot)
                         (availability  generalize-slot)
                         (type          generalize-slot)
                         (place_name    generalize-slot)
                         (place_address generalize-slot)))

(PUTPROP 'store 'slots '((type    generalize-slot)
                         (section generalize-slot)))

(PUTPROP 'public_service_office 'slots '((type    generalize-slot)
                                         (section generalize-slot)))

(PUTPROP 'general_person 'slots '((name        generalize-slot)
                                  (nationality generalize-slot)
                                  (ethnicity   generalize-slot)
                                  (age         number)
                                  (type        generalize-slot)))

(PUTPROP 'prisoner 'slots '((involved_in      generalize-slot)
                            (affiliation      generalize-slot)
```

```
                             (place_imprisoned generalize-slot)))

(PUTPROP 'victim 'slots '((occupa_role generalize-slot)
                          (employer    generalize-slot)
                          (position    generalize-slot)))

(PUTPROP 'diplomat 'slots '((title    generalize-slot)
                            (position generalize-slot)))

(PUTPROP 'release_of_prisoners 'slots '((no_release number)
                                        (name_lst   generalize-slot)))

(PUTPROP 'name_lst 'slots '((person1  generalize-slot)
                            (person2  generalize-slot)
                            (person3  generalize-slot)
                            (person4  generalize-slot)
                            (person5  generalize-slot)
                            (person6  generalize-slot)
                            (person7  generalize-slot)
                            (person8  generalize-slot)
                            (person9  generalize-slot)
                            (person10 generalize-slot)))
```

This is an example of an input frame from the terrorist event story domain. An excerpt of the New York Times news story from which the frame was encoded is included as well.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; PARIS, July 9 [1986] - A powerful bomb ripped through an annex of
;; Paris police headquarters here today, killing a police inspector and
;; wounding at least 22 people, 3 seriously.  The bomb, which the
;; police said contained about 20 pounds of explosives, demolished the
;; fifth and sixth floors of the police annex, which is being used by
;; a special anticrime squad.  No group took responsibility.  ...  The
;; bomb demolished 3,000 square feet of the seven-story police annex,
;; which is on the right bank of the Seine across from the Ile de la
;; Cite, on which police headquarters and the Cathedral of Notre Dame
;; stand.  The annex is at the corner of Quai de Gesvres and Rue
;; St.-Martin, two blocks from City Hall,...  ... the blast occurred at
;; 3:55 P.M.  ...  The police inspector who was killed, Marcel
;; Basdevant, 54 years old, was five months from retirement.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(PUTPROP 'event4 'frame-type 'event)
(FPUT 'event4 'type     'value 'explosion)
(FPUT 'event4 'cause    'value '(cause4 cause))
(FPUT 'event4 'location 'value '(location4 location))
(FPUT 'event4 'time     'value '(time4 time))
(FPUT 'event4 'result   'value '(result4 result))
(FPUT 'event4 'claims   'value '(claim4 claims))
(FPUT 'event4 'warning  'value 'none)

(PUTPROP 'cause4 'frame-type 'cause)
(FPUT 'cause4 'instrument 'value 'bomb)
(FPUT 'cause4 'rev-action 'value 'none)

(PUTPROP 'location4 'frame-type 'location)
```

```
(FPUT 'location4 'country 'value 'france)
(FPUT 'location4 'city    'value 'paris)
(FPUT 'location4 'place   'value '(place4 place))

(PUTPROP 'time4 'frame-type 'time)
(FPUT 'time4 'year       'value 1986)
(FPUT 'time4 'month      'value 'july)
(FPUT 'time4 'day        'value 9)
(FPUT 'time4 'hour       'value 15.92)
(FPUT 'time4 'difference 'value 6)

(PUTPROP 'result4 'frame-type 'result)
(FPUT 'result4 'no_killed         'value 1)
(FPUT 'result4 'killed_victims    'value '(person4_1 general_person))
(FPUT 'result4 'no_hurt_seriously 'value 3)
(FPUT 'result4 'no_hurt           'value 22)
(FPUT 'result4 'no_hostages       'value 0)
(FPUT 'result4 'damage 'value     'extensive)

(PUTPROP 'claim4 'frame-type 'claims) ;; NOTE - FROM A LATER STORY
(FPUT 'claim4 'claimer 'value 'direct_action)
(FPUT 'claim4 'type    'value 'actual)

(PUTPROP 'place4 'frame-type 'place)
(FPUT 'place4 'environ       'value 'public)
(FPUT 'place4 'availability  'value 'open)
(FPUT 'place4 'type          'value 'police_station)
(FPUT 'place4 'place_name    'value 'police_station)
(FPUT 'place4 'place_address 'value 'quai_de_gesvres)

(PUTPROP 'person4_1 'frame-type 'general_person)
(FPUT 'person4_1 'name        'value 'marcel_basdevant)
(FPUT 'person4_1 'nationality 'value 'french)
(FPUT 'person4_1 'age         'value 54)
(FPUT 'person4_1 'type        'value '(victim4_1 victim))

(PUTPROP 'victim4_1 'frame-type 'victim)
(FPUT 'victim4_1 'occupa_role 'value 'police_officer)
(FPUT 'victim4_1 'position    'value 'inspector)
```

Rule Base for Terrorist Event News Stories

The following is a listing of the rule base for the domain of terrorist event news stories. We created this rule base specifically as a testbed for Gemini.

```
(RULE1
   (IF  (isa x? (location))
        (gen x? city beirut))
   (THEN
        (fact area_of_religious_tensions)))

(RULE2
   (IF
        (isa x? (location))
        (has x? country lebanon))
   (THEN
        (fact area_of_religious_tensions)))

(RULE3
   (IF
        (isa x? (result))
```

```
        (grtr-than x? no_killed 20))
   (THEN
        (fact casualties_high)))

(RULE4
   (IF
        (isa x? (result))
        (grtr-than x? no_hurt_seriously 100))
   (THEN
        (fact casualties_high)))

(RULE5
   (IF
        (isa x? (result))
        (grtr-than x? no_hurt 100))
   (THEN
        (fact casualties_high)))

(RULE6
   (IF
        (fact area_of_religious_tensions)
        (isa x? (location))
        (gen x? place religious_place))
   (THEN
        (fact heighten_religious_tensions)))

(RULE7
   (IF
        (fact area_of_religious_tensions)
        (fact casualties_high))
   (THEN
        (fact heighten_religious_tensions)))

(RULE8
   (IF
        (fact area_of_religious_tensions)
        (isa x? (result))
        (has x? damage extensive))
   (THEN
        (fact heighten_religious_tensions)))

(RULE9
   (IF
        (fact heighten_religious_tensions))
   (THEN
        (fact terrorist_event)))

(RULE10
   (IF
        (isa x? (location))
        (has x? country sri_lanka))
   (THEN
        (fact area_of_separatist_fighting)))

(RULE11
   (IF
        (fact area_of_separatist_fighting)
        (isa x? (claims))
        (gen x? claimer separatists))
   (THEN
        (fact separatist_fighting)))

(RULE12
```

```
   (IF
        (fact area_of_separatist_fighting)
        (fact casualties_high))
   (THEN
        (fact assumed_separatist_fighting)))

(RULE13
   (IF
        (fact separatist_fighting))
   (THEN
        (fact terrorist_event)))

(RULE14
   (IF
        (fact assumed_separatist_fighting))
   (THEN
        (fact terrorist_event)))

(RULE15
   (IF
        (isa x? (location))
        (has x? country ireland)
        (isa y? (claims))
        (has y? claimer ira))
   (THEN
        (fact component_of_longstanding_political_violence)))

(RULE16
   (IF
        (fact component_of_longstanding_political_violence))
   (THEN
        (fact terrorist_event)))

(RULE17
   (IF
        (not-isa x? claims none)
        (not-isa x? claims nil))
   (THEN
        (fact specific_threat)))

(RULE18
   (IF
        (isa x? claims none))
   (THEN
        (fact non_specific_threat)))

(RULE19
   (IF
        (isa x? claims nil))
   (THEN
        (fact non_specific_threat)))

(RULE20
   (IF
        (isa x? (target))
        (gen x? target_name authority_person)
        (fact specific_threat))
   (THEN
        (fact specific_threat_towards_authority_individuals)))

(RULE21
   (IF
        (isa x? (target))
```

```
        (gen x? target_name authority_person)
        (fact non_specific_threat))
   (THEN
        (fact general_threat_towards_authority_individuals)))

(RULE22
   (IF
        (fact specific_threat_towards_authority_individuals))
   (THEN
        (fact terrorist_event)))

(RULE23
   (IF
        (fact general_threat_towards_authority_individuals))
   (THEN
        (fact terrorist_event)))

(RULE24
   (IF
        (isa x? (target))
        (gen x? target_name govt_agency))
   (THEN
        (fact general_political_target)))

(RULE25
   (IF
        (isa x? (target))
        (gen x? target_name political_group))
   (THEN
        (fact general_political_target)))

(RULE26
   (IF
        (isa x? (location place))
        (gen x? type government_place))
   (THEN
        (fact general_political_target)))

(RULE27
   (IF
        (isa x? (location place))
        (gen x? type political_place))
   (THEN
        (fact general_political_target)))

(RULE28
   (IF
        (isa x? (location place))
        (gen x? place_name government_place))
   (THEN
        (fact general_political_target)))

(RULE29
   (IF
        (isa x? (location place))
        (gen x? place_name political_place))
   (THEN
        (fact general_political_target)))

(RULE30
   (IF
        (isa x? (location))
        (gen x? place government_place))
```

```
   (THEN
        (fact general_political_target)))

(RULE31
   (IF
        (isa x? (location))
        (gen x? place political_place))
   (THEN
        (fact general_political_target)))

(RULE32
   (IF
        (isa x? (cause))
        (gen x? instrument explosives))
   (THEN
        (fact massive_weaponry)))

(RULE33
   (IF
        (isa x? (result))
        (not-has x? damage none)
        (not-has x? damage nil))
   (THEN
        (fact damage_sustained)))

(RULE34
   (IF
        (fact general_political_target)
        (fact massive_weaponry))
   (THEN
        (fact violence_against_political_structure)))

(RULE35
   (IF
        (fact general_political_target)
        (fact damage_sustained))
   (THEN
        (fact violence_against_political_structure)))

(RULE36
   (IF
        (fact violence_against_political_structure))
   (THEN
        (fact terrorist_event)))

(RULE37
   (IF
        (isa x? (location))
        (gen x? place public_place)
        (not-gen x? place political_place)
        (not-gen x? place government_place))
   (THEN
        (fact general_public_threatened)))

(RULE38
   (IF
        (isa x? (location place))
        (gen x? type public_place)
        (not-gen x? type political_place)
        (not-gen x? type government_place))
   (THEN
        (fact general_public_threatened)))
```

```
(RULE39
   (IF
        (fact general_public_threatened)
        (isa x? target nil)
        (isa x? type explosion)
        (isa y? (result))
        (not-grtr-than y? no_killed 2)
        (not-grtr-than y? no_hurt_seriously 2)
        (not-grtr-than y? no_hurt 2)
        (not-has y? damage none)
        (not-has y? damage nil))
   (THEN
        (fact location_organization_or_company_threatened)))

(RULE40
   (IF
        (fact location_organization_or_company_threatened))
   (THEN
        (fact terrorist_event)))

(RULE41
   (IF
        (isa x? type explosion))
   (THEN
        (fact massive_force)))

(RULE42
   (IF
        (isa x? type arson))
   (THEN
        (fact massive_force)))

(RULE43
   (IF
        (isa x? type attack))
   (THEN
        (fact massive_force)))

(RULE44
   (IF
        (isa x? warning none))
   (THEN
        (fact not_significant_warning)))

(RULE45
   (IF
        (isa x? warning nil))
   (THEN
        (fact not_significant_warning)))

(RULE46
   (IF
        (not-isa x? warning none)
        (not-isa x? warning nil))
   (THEN
        (fact some_warning)))

(RULE47
   (IF
        (isa x? (result))
        (grtr-than x? no_killed 2))
   (THEN
        (fact non_trivial_casualties)))
```

```
(RULE48
   (IF
        (isa x? (result))
        (grtr-than x? no_hurt_seriously 2))
   (THEN
        (fact non_trivial_casualties)))

(RULE49
   (IF
        (isa x? (result))
        (grtr-than x? no_hurt 5))
   (THEN
        (fact non_trivial_casualties)))

(RULE50
   (IF
        (isa x? (location place))
        (has x? environ public)
        (has x? availability open)
        (fact not_significant_warning))
   (THEN
        (fact crowded_target))
   (CONTEXT ((time) hour business-hours)))

(RULE51
   (IF
        (fact some_warning)
        (fact non_trivial_casualties))
   (THEN
        (fact crowded_target)))

(RULE52
   (IF
        (fact massive_force)
        (fact crowded_target)
        (isa x? target nil))
   (THEN
        (fact attack_thru_fear_of_genl_public)))

(RULE53
   (IF
        (fact attack_thru_fear_of_genl_public))
   (THEN
        (fact terrorist_event)))
```

Appendix C

A Complete Gemini Trace

C.1. Introduction

This appendix traces a rule learning session in Gemini. This trace was generated by running an actual Gemini session. We have included the functions executed in Gemini (written in CommonLisp). However, we have also added our own notes to explain each step.

The domain used in this session is the network fault diagnosis domain. The trace will demonstrate rule induction when the following rule is missing from the domain theory:

```
(RULE3
      (IF
          (isa x? td-the-user-is-on-a-TRN? t)
          (fact looks-like-source-prob))
      (THEN
          (fact looks-like-source-prob-TRN)))
```

The complete rule base for this domain is given in Appendix B. Chapter 5 describes test runs characterizing the performance of various heuristics when learning this rule.

C.2. Setting Up the Learning Session

Say that Gemini has been operating for a short time. During that time it has processed 3 inputs. Each input to Gemini is a pair consisting of the name of a concept (in this domain, a diagnosis) and an instance of that concept. The instance is represented as a frame. Say that the 3 inputs to Gemini thus far have been:

1. <looks-like-dest-prob, FAULT4>
2. <looks-like-dest-prob, FAULT5>
3. <poss-remote-copper-rptr-prob, FAULT30>,

where the frames for the faults are as follows:

```
complaint-user-cant-connect?      t      FAULT4 (looks-like-dest-prob)
cond-is-it-a-single-remote-MAU?   f
cond-rptrs-are-in-fault-domain?   f
cond-segnet-usr-can-initialize?   t
cond-TD-gets-PC-init-failure?     f
td-all-nodes-gone-fault-domain?   f
td-can-ping-nodes-but-not-user?   f
td-can-ping-source-net?           t
td-can-user-reach-another-net?    t
td-local-fbr-rptr-low-lite-on?    f
td-local-fbr-rptr-low-signal?     f
td-local-fbr-rptr-pwr-lite-off?   f
td-local-loopback-test?           t
td-remot-fbr-rptr-pwr-lite-off?   f
td-remote-fbr-rptr-low-lite-on?   f
td-remote-loopback-test?          f
td-rt-console-hung/no-msg?        f
td-rtr-console-has-error-msg?     f
td-the-user-is-on-a-TRN?          f
td-the-user-is-on-an-Ethernet?    t
td-user-cant-communicate-at-all   f
td-user-says-cant-initialize?     f
td-usr-gets-init-retry-failure?   f


complaint-user-cant-connect?      t      FAULT5 (looks-like-dest-prob)
cond-is-it-a-single-remote-MAU?   f
cond-rptrs-are-in-fault-domain?   f
td-all-nodes-gone-fault-domain?   f
td-can-ping-nodes-but-not-user?   f
td-can-ping-rtr-from-MAU?         t
td-can-ping-source-net?           t
td-can-user-reach-another-net?    t
td-local-fbr-rptr-low-lite-on?    f
td-local-fbr-rptr-low-signal?     f
td-local-fbr-rptr-pwr-lite-off?   f
td-local-loopback-test?           t
td-remot-fbr-rptr-pwr-lite-off?   f
td-remote-fbr-rptr-low-lite-on?   f
td-remote-loopback-test?          t
td-the-user-is-on-a-TRN?          f
td-the-user-is-on-an-Ethernet?    t
td-user-cant-communicate-at-all   f
td-user-says-cant-initialize?     f
td-user-wall-connection-OK?       t


complaint-user-cant-connect?      t      FAULT30
cond-is-it-a-single-remote-MAU?   f     (poss-remote-copper-rptr-prob)
cond-rptrs-are-in-fault-domain?   t
cond-TD-gets-PC-init-failure?     f
cond-tgt-node-responds-to-ping?   t
td-all-nodes-gone-fault-domain?   f
td-can-ping-nodes-but-not-user?   f
td-can-ping-rtr-fm-Edinboro?      f
td-can-ping-rtr-from-MAU?         t
td-can-ping-source-net?           f
td-can-user-reach-another-net?    f
td-local-fbr-rptr-low-lite-on?    f
td-local-fbr-rptr-low-signal?     f
td-local-fbr-rptr-pwr-lite-off?   f
```

```
td-local-loopback-test?           f
td-remot-fbr-rptr-pwr-lite-off?   t
td-remote-fbr-rptr-low-lite-on?   f
td-remote-loopback-test?          f
td-rt-console-hung/no-msg?        f
td-rtr-console-has-error-msg?     f
td-the-user-is-on-a-TRN?          t
td-user-cant-communicate-at-all   f
td-user-says-cant-initialize?     f
td-user-wall-connection-OK?       t
td-usr-gets-init-retry-failure?   f
```

For each input, Gemini has generated an explanation describing why the frame is an instance of the goal concept. The explanations generated by Gemini for the above 3 inputs are as follows:

```
> (explain 'fault4 'looks-like-dest-prob)
(((T))
  ((RULE2
     (IF (ISA FAULT4 COMPLAINT-USER-CANT-CONNECT? T)
         (ISA FAULT4 TD-CAN-USER-REACH-ANOTHER-NET? T))
     (THEN (FACT LOOKS-LIKE-DEST-PROB)))))

> (explain 'fault5 'looks-like-dest-prob)
(((T))
  ((RULE2
     (IF (ISA FAULT5 COMPLAINT-USER-CANT-CONNECT? T)
         (ISA FAULT5 TD-CAN-USER-REACH-ANOTHER-NET? T))
     (THEN (FACT LOOKS-LIKE-DEST-PROB)))))

> (explain 'fault30 'poss-remote-copper-rptr-prob)
((((RULE?????
     (IF ?????????
     (THEN (FACT LOOKS-LIKE-SOURCE-PROB-TRN))))
  ((RULE10
     (IF (ISA FAULT30 TD-CAN-PING-SOURCE-NET? F)
         (FACT LOOKS-LIKE-SOURCE-PROB-TRN))
     (THEN (FACT POSS-FAILURE-SRC-TRN))))
  ((RULE18
     (IF (FACT POSS-FAILURE-SRC-TRN))
     (THEN (FACT POSS-REMOTE-FAILURE))))
  ((RULE52
     (IF (FACT POSS-REMOTE-FAILURE))
     (THEN (FACT POSS-REMOTE-TRN-PROB))))
  ((RULE53
     (IF (ISA FAULT30 COND-RPTRS-ARE-IN-FAULT-DOMAIN? T)
         (FACT POSS-REMOTE-TRN-PROB))
     (THEN (FACT POSS-REMOTE-COPPER-RPTR-PROB)))))
```

Note that the explanation for FAULT30 is incomplete as it requires the missing rule. If the rule had been in the knowledge base, the explanation would have been:

```
> (explain 'fault30 'poss-remote-copper-rptr-prob)
(((T))
  ((RULE1
```

```
    (IF (ISA FAULT30 COMPLAINT-USER-CANT-CONNECT? T)
        (ISA FAULT30 COND-TGT-NODE-RESPONDS-TO-PING? T)
        (ISA FAULT30 TD-CAN-USER-REACH-ANOTHER-NET? F))
    (THEN (FACT LOOKS-LIKE-SOURCE-PROB))))
 ((RULE3
    (IF (ISA FAULT30 TD-THE-USER-IS-ON-A-TRN? T)
        (FACT LOOKS-LIKE-SOURCE-PROB))
    (THEN (FACT LOOKS-LIKE-SOURCE-PROB-TRN))))
 ((RULE10
    (IF (ISA FAULT30 TD-CAN-PING-SOURCE-NET? F)
        (FACT LOOKS-LIKE-SOURCE-PROB-TRN))
    (THEN (FACT POSS-FAILURE-SRC-TRN))))
 ((RULE18
    (IF (FACT POSS-FAILURE-SRC-TRN))
    (THEN (FACT POSS-REMOTE-FAILURE))))
 ((RULE52
    (IF (FACT POSS-REMOTE-FAILURE))
    (THEN (FACT POSS-REMOTE-TRN-PROB))))
 ((RULE53
    (IF (ISA FAULT30 COND-RPTRS-ARE-IN-FAULT-DOMAIN? T)
        (FACT POSS-REMOTE-TRN-PROB))
    (THEN (FACT POSS-REMOTE-COPPER-RPTR-PROB)))))
```

Note that the partial explanation for FAULT30 generated by Gemini is consistent with the partial explanation obtained by starting with the complete explanation and removing by hand all parts of the explanation tree below the missing subgoal.

Now suppose that it was not immediately necessary to complete the explanation of FAULT30 (i.e., that it was acceptable for Gemini to simply proceed in its operation). It will continue to process inputs and to generate explanations for them. Before continuing, however, Gemini will have stored the inputs already seen into its generalization-based memory. Inputs are stored there as they might be of value for rule induction later. For the 3 inputs above, Gemini created a GBM as follows:

```
> (gbm '(fault4 fault5 fault30))

Creating THE-GENERALIZATION-HIERARCHY with the following frames:
(FAULT4 FAULT5 FAULT30)
<level 2 >
(1 . FAULT201)  (GEN-LIST NIL)  (SPEC-LIST ((1 . 1) (1 . 2)))
<level 1 >
(1 . FAULT4)  (GEN-LIST ((2 . 1)))  (SPEC-LIST NIL)
(2 . FAULT5)  (GEN-LIST ((2 . 1)))  (SPEC-LIST NIL)
(3 . FAULT30)  (GEN-LIST NIL)  (SPEC-LIST NIL)
```

Gemini found FAULT4 and FAULT5 to be similar and has grouped them together as specializations of a new (generalized) fault called FAULT201. FAULT30 was stored by itself, separate from the other 2.

C.3. Induction of a New Rule

Now say that a new input enters Gemini: <poss-remote-mau-prob, FAULT80>. Gemini must try to generate an explanation showing that FAULT80 is an instance of the concept *poss-remote-mau-prob*. FAULT80 is represented as follows:

```
complaint-user-cant-connect?      t      FAULT80 (poss-remote-MAU-prob)
cond-is-it-a-single-remote-MAU?   t
cond-rptrs-are-in-fault-domain?   f
cond-TD-gets-PC-init-failure?     f
cond-tgt-node-responds-to-ping?   t
td-all-nodes-gone-fault-domain?   t
td-can-ping-nodes-but-not-user?   f
td-can-ping-rtr-fm-Edinboro?      t
td-can-ping-rtr-from-MAU?         f
td-can-ping-source-net?           f
td-can-user-reach-another-net?    f
td-local-fbr-rptr-low-lite-on?    f
td-local-fbr-rptr-low-signal?     f
td-local-fbr-rptr-pwr-lite-off?   f
td-remot-fbr-rptr-pwr-lite-off?   f
td-remote-fbr-rptr-low-lite-on?   f
td-rt-console-hung/no-msg?        f
td-the-user-is-on-a-TRN?          t
td-the-user-is-on-an-Ethernet?    f
td-user-cant-communicate-at-all   f
td-user-wall-connection-OK?       t
```

Gemini generates the following explanation for this input:

```
> (explain 'fault80 'poss-remote-mau-prob)
((((RULE?????
     (IF ?????????
     (THEN (FACT LOOKS-LIKE-SOURCE-PROB-TRN))))
  ((RULE10
     (IF (ISA FAULT80 TD-CAN-PING-SOURCE-NET? F)
         (FACT LOOKS-LIKE-SOURCE-PROB-TRN))
     (THEN (FACT POSS-FAILURE-SRC-TRN))))
  ((RULE18
     (IF (FACT POSS-FAILURE-SRC-TRN))
     (THEN (FACT POSS-REMOTE-FAILURE))))
  ((RULE52
     (IF (FACT POSS-REMOTE-FAILURE))
     (THEN (FACT POSS-REMOTE-TRN-PROB))))
  ((RULE56
     (IF (ISA FAULT80 COND-IS-IT-A-SINGLE-REMOTE-MAU? T)
         (ISA FAULT80 TD-ALL-NODES-GONE-FAULT-DOMAIN? T)
         (FACT POSS-REMOTE-TRN-PROB))
     (THEN (FACT POSS-REMOTE-MAU-PROB)))))
```

Like the explanation generated for FAULT30 earlier, this one is incomplete as it requires the missing rule. If the missing rule had been present, the complete explanation would have been:

```
> (explain 'fault80 'poss-remote-mau-prob)
(((T))
  ((RULE1
     (IF (ISA FAULT80 COMPLAINT-USER-CANT-CONNECT? T)
         (ISA FAULT80 COND-TGT-NODE-RESPONDS-TO-PING? T)
         (ISA FAULT80 TD-CAN-USER-REACH-ANOTHER-NET? F))
     (THEN (FACT LOOKS-LIKE-SOURCE-PROB))))
  ((RULE3
     (IF (ISA FAULT80 TD-THE-USER-IS-ON-A-TRN? T)
         (FACT LOOKS-LIKE-SOURCE-PROB))
     (THEN (FACT LOOKS-LIKE-SOURCE-PROB-TRN))))
  ((RULE10
     (IF (ISA FAULT80 TD-CAN-PING-SOURCE-NET? F)
         (FACT LOOKS-LIKE-SOURCE-PROB-TRN))
     (THEN (FACT POSS-FAILURE-SRC-TRN))))
  ((RULE18
     (IF (FACT POSS-FAILURE-SRC-TRN))
     (THEN (FACT POSS-REMOTE-FAILURE))))
  ((RULE52
     (IF (FACT POSS-REMOTE-FAILURE))
     (THEN (FACT POSS-REMOTE-TRN-PROB))))
  ((RULE56
     (IF (ISA FAULT80 COND-IS-IT-A-SINGLE-REMOTE-MAU? T)
         (ISA FAULT80 TD-ALL-NODES-GONE-FAULT-DOMAIN? T)
         (FACT POSS-REMOTE-TRN-PROB))
     (THEN (FACT POSS-REMOTE-MAU-PROB)))))
```

Say that this time the complete explanation is required immediately. Thus Gemini must begin rule induction. Before inducing a new rule, Gemini places the new input in the GBM. This serves 2 purposes. It makes the input available for later retrieval, should it become necessary. It also gives Gemini an indication of what other (previously seen) inputs are most closely related to the current one.

```
> (gbm '(fault80))

Adding into THE-GENERALIZATION-HIERARCHY the following frames:
(FAULT80)
<level 2 >
(1 . FAULT201)  (GEN-LIST NIL)  (SPEC-LIST ((1 . 1) (1 . 2)))
<level 1 >
(1 . FAULT4)  (GEN-LIST ((2 . 1)))  (SPEC-LIST NIL)
(2 . FAULT5)  (GEN-LIST ((2 . 1)))  (SPEC-LIST NIL)
(3 . FAULT30)  (GEN-LIST NIL)  (SPEC-LIST NIL)
(4 . FAULT80)  (GEN-LIST NIL)  (SPEC-LIST NIL)
```

Gemini places the new input into memory, without connecting it to any other instances, as they are not sufficiently related.

Now Gemini can begin the actual rule induction. Gemini learns new rules by invoking contextual heuristics that make use of explanatory information. The heuristics select the instances from which the new rule is to be induced, and indicate the manner in which they are to be used. All instance selections are made from the GBM.

First Gemini begins fairly conservatively. It looks for other instances in the GBM that require a rule for the same unproven subgoal, *looks-like-source-prob-trn*. Furthermore, it requires that the retrieved instances be very similar to the current trigger instance both with respect to its frame representation and its explanation. This allows Gemini to focus on other positive instances that can presumably be covered by a simple conjunctive definition.

```
> (find-i-set 'fault80 10)

<related cases>

***** WARNING
INSUFFICIENT NUMBER OF RELATED CASES RETURNED

(FAULT80)
```

In this case, there are no closely related instances other than FAULT80 itself. If there had been, they would have been stored in the GBM as siblings or ‘‘cousins’’ of FAULT80. Here a parameter has been set by a user indicating that 10 instances will generally be enough for this step. It would often be higher, but was set this way due to the small size of this trace.

From the closely related instances (in this case just the trigger frame itself), Gemini creates a new frame that is a minimal generalization of the instance features. This frame represents the features that are being considered for the left-hand-side (lhs) of the new rule:

```
> (action-0 '(fault80) 'fault80)
FAULT211
> (frame-to-rule 'fault211 missing)

      ;;; NOTE: missing = looks-like-source-prob-trn

(RULE60
  (IF (ISA ?A COMPLAINT-USER-CANT-CONNECT? T)
      (ISA ?A COND-IS-IT-A-SINGLE-REMOTE-MAU? T)
      (ISA ?A COND-RPTRS-ARE-IN-FAULT-DOMAIN? F)
      (ISA ?A COND-TD-GETS-PC-INIT-FAILURE? F)
      (ISA ?A COND-TGT-NODE-RESPONDS-TO-PING? T)
      (ISA ?A TD-ALL-NODES-GONE-FAULT-DOMAIN? T)
      (ISA ?A TD-CAN-PING-NODES-BUT-NOT-USER? F)
      (ISA ?A TD-CAN-PING-RTR-FM-EDINBORO? T)
```

```
      (ISA ?A TD-CAN-PING-RTR-FROM-MAU? F)
      (ISA ?A TD-CAN-PING-SOURCE-NET? F)
      (ISA ?A TD-CAN-USER-REACH-ANOTHER-NET? F)
      (ISA ?A TD-LOCAL-FBR-RPTR-LOW-LITE-ON? F)
      (ISA ?A TD-LOCAL-FBR-RPTR-LOW-SIGNAL? F)
      (ISA ?A TD-LOCAL-FBR-RPTR-PWR-LITE-OFF? F)
      (ISA ?A TD-REMOT-FBR-RPTR-PWR-LITE-OFF? F)
      (ISA ?A TD-REMOTE-FBR-RPTR-LOW-LITE-ON? F)
      (ISA ?A TD-RT-CONSOLE-HUNG/NO-MSG? F)
      (ISA ?A TD-THE-USER-IS-ON-A-TRN? T)
      (ISA ?A TD-THE-USER-IS-ON-AN-ETHERNET? F)
      (ISA ?A TD-USER-CANT-COMMUNICATE-AT-ALL F)
      (ISA ?A TD-USER-WALL-CONNECTION-OK? T))
  (THEN (FACT LOOKS-LIKE-SOURCE-PROB-TRN))
```

A least generalization is created by applying the actions of test heuristic 0, described in Chapter 5. We have generated a new rule from the new frame to show the features that are currently under consideration for the left-hand-side of the new rule. They are the features of FAULT80.

Next Gemini applies a number of contextual heuristics to generalize the left-hand-side of the new rule, so that it will be more widely applicable. First it applies test heuristic 1, which is:

```
Test Heuristic 1: ‘‘Least generalization from liberal instances.’’

IF
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is SAME-AS past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is SAME-AS past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  KEEP features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

Gemini searches the GBM to find earlier instances that match the conditions of the heuristic:

```
> (lhs-heur-1 missing ’fault80 memory)
(FAULT30)
```

And then it applies the actions:

```
> (action-1 ’fault211 ’fault80 ’(fault30))
FAULT211
> (frame-to-rule ’fault211 missing)
(RULE61
  (IF (ISA ?A COMPLAINT-USER-CANT-CONNECT? T)
      (ISA ?A COND-TD-GETS-PC-INIT-FAILURE? F)
```

```
      (ISA ?A COND-TGT-NODE-RESPONDS-TO-PING? T)
      (ISA ?A TD-CAN-PING-NODES-BUT-NOT-USER? F)
      (ISA ?A TD-CAN-PING-SOURCE-NET? F)
      (ISA ?A TD-CAN-USER-REACH-ANOTHER-NET? F)
      (ISA ?A TD-LOCAL-FBR-RPTR-LOW-LITE-ON? F)
      (ISA ?A TD-LOCAL-FBR-RPTR-LOW-SIGNAL? F)
      (ISA ?A TD-LOCAL-FBR-RPTR-PWR-LITE-OFF? F)
      (ISA ?A TD-REMOTE-FBR-RPTR-LOW-LITE-ON? F)
      (ISA ?A TD-RT-CONSOLE-HUNG/NO-MSG? F)
      (ISA ?A TD-THE-USER-IS-ON-A-TRN? T)
      (ISA ?A TD-USER-CANT-COMMUNICATE-AT-ALL F)
      (ISA ?A TD-USER-WALL-CONNECTION-OK? T))
  (THEN LOOKS-LIKE-SOURCE-PROB-TRN))
```

The temporary frame containing the features under consideration for the new rule has been modified to include only those features also appearing in the instance retrieved, FAULT30. The left-hand-side has been generalized considerably, containing 7 fewer conditions that must be met.

Next Gemini applies a second contextual heuristic that searches for very *un*-related instances that it treats as negative examples of the missing subgoal:

```
Test Heuristic 7: ''Features irrelevant - explanation's
                    the determining factor.''

IF
  current example x is SAME-AS past example y in EXAMPLE-FEATURES AND
  current example x is DIFF-FROM past example y in GOAL-CONCEPT AND
  current example x is DIFF-FROM past example y in SUBGOALS AND
  current example x is DIFF-FROM past example y in LEAVES AND
  current example x is DIFF-FROM past example y in UNPROVEN-SUBGOAL AND
  current example x is DIFF-FROM past example y in RULES-USED
THEN
  ELIM features in rule that are the same as those covered by
     EXAMPLE-FEATURES in example y.
```

As for the previous heuristic, Gemini first searches the GBM for instances that meet the conditions of the heuristic:

```
> (lhs-heur-7 missing 'fault80 memory)
(FAULT5 FAULT4)
```

And then it applies the actions of the heuristic to revise the current potential rule:

```
> (action-7 'fault211 'fault80 '(fault4 fault5))
FAULT211
> (frame-to-rule 'fault211 missing)
(RULE62
  (IF (ISA ?A COND-TGT-NODE-RESPONDS-TO-PING? T)
      (ISA ?A TD-CAN-PING-SOURCE-NET? F)
      (ISA ?A TD-CAN-USER-REACH-ANOTHER-NET? F)
```

```
      (ISA ?A TD-THE-USER-IS-ON-A-TRN? T))
  (THEN LOOKS-LIKE-SOURCE-PROB-TRN))
```

With the application of test heuristic 7, the new rule has be generalized considerably.

Finally, Gemini applies a third heuristic, test heuristic 9, which is quite different from the earlier ones. It does not retrieve any instances, but rather generalizes the new rule by removing features already appearing in the partial explanation of the trigger instance (FAULT80):

```
> (action-9 'fault211 'fault80 nil)
FAULT211
> (frame-to-rule 'fault211 missing)
(RULE63
  (IF (ISA ?A COND-TGT-NODE-RESPONDS-TO-PING? T)
      (ISA ?A TD-CAN-USER-REACH-ANOTHER-NET? F)
      (ISA ?A TD-THE-USER-IS-ON-A-TRN? T))
  (THEN LOOKS-LIKE-SOURCE-PROB-TRN))
```

At this point say that Gemini has made use of all the heuristics known to be effective for this domain. It has a new rule, but the rule refers only to input features. It does not refer to any subgoals in its left-hand-side. Since the rule might be missing ‘‘from the middle’’ of an explanation, Gemini attempts to infer a subgoal from the lhs features.

```
> (inferences-from-example 'fault211)
FAULT211
> (frame-to-rule 'fault211 missing)
(RULE64
  (IF (ISA ?A COND-TGT-NODE-RESPONDS-TO-PING? T)
      (ISA ?A TD-CAN-USER-REACH-ANOTHER-NET? F)
      (ISA ?A TD-THE-USER-IS-ON-A-TRN? T))
  (THEN LOOKS-LIKE-SOURCE-PROB-TRN))
```

In this case, attempting to replace some set of input features with a subgoal has resulted in no change. We can compare the learned rule with the one missing from the domain theory:

```
(RULE3
      (IF
          (isa x? td-the-user-is-on-a-TRN? t)
          (fact looks-like-source-prob))
      (THEN
          (fact looks-like-source-prob-TRN)))
```

The 2 are very similar. Though somewhat different, the rule created by Gemini is, in fact, quite possibly better than the original. To understand why, we analyze the rules. In the original rule, RULE3, the lhs has a condition that requires that the subgoal *looks-like-source-prob* be inferred. The only rule in the domain theory that infers *looks-like source-prob* is RULE1:

```
(RULE1
      (IF
          (isa x? complaint-user-cant-connect? t)
          (isa x? cond-tgt-node-responds-to-ping? t)
          (isa x? td-can-user-reach-another-net? f))
      (THEN
          (fact looks-like-source-prob)))
```

Comparing this rule to the one learned by Gemini, we see that the only feature missing in order to make the inference is *complaint-user-cant-connect?*. But this feature is essentially irrelevant. It simply indicates that there is a fault situation. A fault in this domain is signaled when a user cannot make a connection. Therefore, the presence of the feature in any rule provides no additional information. If RULE1 were modified to exclude this unnecessary feature, the rule learned by Gemini would be:

```
> (rule-set-up)
T
> (inferences-from-example ’fault211)
FAULT211
> (frame-to-rule ’fault211 missing)
(RULE65 (IF (ISA ?A TD-THE-USER-IS-ON-A-TRN? T)
            (FACT LOOKS-LIKE-SOURCE-PROB))
        (THEN LOOKS-LIKE-SOURCE-PROB-TRN))
```

Note that Gemini was able to learn this rule from only 4 instances, including the instance that triggered the induction.

Table of Contents

List of Figures