A GENERAL EXPLANATION-BASED LEARNING MECHANISM AND ITS APPLICATION TO NARRATIVE UNDERSTANDING

BY

RAYMOND JOSEPH MOONEY

B.S., University of Illinois at Urbana-Champaign, 1983
M.S., University of Illinois at Urbana-Champaign, 1985

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1988

Urbana, Illinois

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

JANUARY 1988

WE HEREBY RECOMMEND THAT THE THESIS BY

RAYMOND JOSEPH MOONEY

ENTITLED A GENERAL EXPLANATION-BASED LEARNING MECHANISM AND ITS APPLICATION TO NARRATIVE UNDERSTANDING

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Director of Thesis Research

Head of Department

Committee on Final Examination†

Chairperson

William F. Brewer

† Required for doctor's degree but not for master's.

O-517

A GENERAL EXPLANATION-BASED LEARNING MECHANISM AND ITS APPLICATION TO NARRATIVE UNDERSTANDING

Raymond Joseph Mooney, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1988
Gerald Francis DeJong, Advisor

Explanation-based learning (EBL) is a learning method which uses existing knowledge of the domain to construct an explanation for why a specific example is a member of a concept or why a specific combination of actions achieves a goal. This explanation is then generalized in an analytical manner in order to produce a general concept description or plan schema. Although a number of exploratory EBL systems which operate in particular domains have previously been constructed, recent research in this area has lead to the development of general mechanisms which can perform explanation-based learning in a wide variety of domains.

This thesis describes a general EBL mechanism, EGGS, which can make use of declarative knowledge stored in the form of horn clauses, rewrite rules, or STRIPS operators. Numerous examples are presented illustrating its application to a wide variety of domains, including "blocks world" planning, logic circuit design, artifact recognition, and various forms of mathematical problem solving. The system is shown to improve its performance in each of these domains.

EGGS has been most thoroughly tested as a component of a narrative understanding system, GENESIS, which improves its own performance through learning. GENESIS processes short English narratives and constructs explanations for characters' intentional behavior. When the system detects that a character has achieved an important goal by combining actions in an unfamiliar way, EGGS is used to generalize the specific explanation for how the goal was achieved into a general plan schema. The resulting schema is then retained by the system and indexed into its existing knowledge-base. This schema can then be used to process narratives which were previously beyond the system's capabilities. The thesis also discusses GENESIS' ability to learn meanings for words related to its learned schemata and reviews several recent psychological experiments which demonstrate that GENESIS can be productively interpreted as a cognitive model of certain types of human learning.

DEDICATION

To the memory of my parents.

If only they had been fortunate enough to witness the completion of an educational endeavor which they always supported.

ACKNOWLEDGEMENTS

I would like to thank:

Gerald DeJong, my advisor, for initiating and directing this research, for innumerable helpful comments, and for leading a stimulating and friendly research group in which it has been a pleasure to work.

William Brewer, for serving on my final committee and for his interest and direction in an enjoyable and enlightening interdisciplinary project in cognitive science.

Gregg Collins, Ken Forbus, and Alan Frisch for also serving on my final committee and providing helpful comments.

Woo-Kyoung Ahn for being a great cognitive science collaborator as well as a good friend.

The EBL group at CSL for many interesting meetings and lunchtime discussions and for being friends as well as colleagues. Particularly to:

Jude Shavlik for many interesting and helpful discussions about EBL, machine learning, and AI in general.

Steve Chien for useful and interesting discussions about nonlinear planning, order generalization, and intractability.

Scott Bennett for initial discussions and work on the general generalizer.

Paul O'Rorke for early discussions about explanation generalization.

Alberto Segre for early discussions on understanding.

Shankar Rajamoney for interesting discussions about imperfect theories.

Brad Whitehall for being a great office-mate and for many interesting discussions.

Bob Stepp and his other students for bringing SDBL to CSL.

Smadar Kedar-Cabelli, Haym Hirsh, Prasad Tadepalli, Steve Minton, and Rich Keller for interesting and helpful discussions on explanation-based generalization.

Dedre Gentner for helpful information on developmental work in word learning and for running interesting cognitive science seminars.

This research was conducted at the Coordinated Science Laboratory of the University of Illinois and was supported by the Air Force Office of Scientific Research under grant F49620-82-K-0009, by the National Science Foundation under grant NSF-IST-83-17889, by the Office of Naval Research under grant N-00014-86-K-0309, and by a University of Illinois Cognitive Science/AI fellowship.

PREFACE

Artificial Intelligence is a young science whose methodology is still developing. Their currently exist a number of relatively distinct methodological perspectives or *paradigms* [Kuhn70] in AI research. Hall and Kibler [Hall85], have proposed the following taxonomy of methodological perspectives in AI:[1]

- Performance AI Oriented towards constructing systems with impressive levels of performance in a particular domain.
- Constructive AI Oriented towards learning general principles of intelligence by designing, constructing, and experimenting with systems operating in complex domains.
- Formal AI Oriented towards formal specification of general problems and mathematical proofs of the correctness and computational complexity of algorithms for solving these problems.
- Speculative AI Oriented towards developing a theory of human cognition by introspection and construction of AI systems. Interested in global, ecologically valid tasks which do not allow for detailed empirical validation.
- Empirical AI Oriented towards developing a theory of human cognition by constructing AI systems that closely model empirical data. Interested in well-defined problems which allow for detailed empirical validation.

The research presented in this thesis does not strictly adhere to any one of these perspectives and cuts across the boundaries of many of them.

Since it has involved constructing and experimenting with a large software system, the research presented here can be seen as adhering to the constructive approach. In this vein, the thesis attempts to provide a detailed description of the system and presents numerous concrete examples of its operation and empirical data on its performance.

However, this thesis also presents precise descriptions of the underlying algorithms used in the system and in several cases mathematically analyzes the computational time complexity of these algorithms. Consequently, it can also be seen as exhibiting aspects of the formal approach.

[1]This is not meant to imply that Hall and Kibler claim that all research in AI can be pigeonholed into one of these categories. In fact, they explicitly acknowledge the "...tendency on the part of researchers and issues to shift quite freely between perspectives..." (p. 173).

Also characteristic of the formal approach is the use of predicate calculus as an underlying representation language.

This research also maintains an interest in human cognition. The development of the system was motivated and guided by intuitions about human learning. Characteristic of the speculative approach, the task of narrative understanding is used as the primary domain of application. Narrative understanding is a more *global* [Lehnert84] or ecologically valid task than well-defined tasks such as the eight-puzzle.

However, the psychological plausibility of the system is not supported only by intuition and anecdotal evidence. Empirical evidence from psychological experiments is used to support the claim that the system exhibits behavior similar to that found in human subjects. Therefore, this research also has features of the empirical approach.

In summary, this thesis attempts to illustrate that a uniform and coherent piece of AI research can productively employ aspects of many of the current methodological perspectives in artificial intelligence. It is the author's sincere hope that more researchers in AI will learn to appreciate the advantages of the different approaches instead of slavishly adhering to a particular methodological mind-set.

TABLE OF CONTENTS

CHAPTER

CHAPTER

CHAPTER

CHAPTER 1

INTRODUCTION

Over the past several years, research in machine learning has included an ever increasing number of projects in the area of *explanation-based learning* (EBL) [DeJong86b, Mitchell86]. Previously, research in machine learning was predominantly concerned with what have been called *similarity-based* [Lebowitz86] or *empirical* [Langley86] learning methods. Prototypically, a similarity-based learning (SBL) method involves determining a concept definition by examining a large number of examples and counter-examples of a concept and searching for a simple description that includes all of the examples while excluding all of the counter-examples (see [Dietterich82, Dietterich83, Mitchell82] for overviews of such methods). A critique of these methods as well as numerous replies to this critique can be found in [Schank86b]. In contrast to this approach, an explanation-based method uses its existing knowledge of the domain to explain why a single example is a member of a concept and then analytically generalizes this explanation to determine a description of the general concept. A discussion of the advantages and disadvantages of this approach can be found in [Mitchell86] and a review of work in EBL is presented in [Ellman87]. Some interesting remarks on the relation between SBL and EBL are also presented in [Dietterich86].

This thesis presents a general domain-independent explanation-based learning mechanism that has been tested on numerous examples from various domains. In addition, it discusses how this learning system has been integrated with a narrative understanding mechanism to form a complete system that learns new plan schemata from English narratives and uses these schemata to improve its understanding ability. Some recent psychological experiments that indicate people can learn a schema by explaining and generalizing a single instance presented in a narrative are also briefly reviewed. Therefore, an overall claim of the entire thesis is that a general mechanism for explanation-based learning can be used to improve the performance of a narrative understanding system in a psychologically plausible manner.

1.1. EGGS: A General Explanation-Based Learning Mechanism

EGGS is a general EBL system that has been tested on a variety of examples from the literature on explanation-based learning. Many of these examples were originally used to demonstrate earlier, more domain-dependent systems such as STRIPS [Fikes72], LEX2 [Mitchell83], CUPS [Winston83], MA [O'Rorke84], and LEAP [Mitchell85]. The examples from these systems come from a number of different domains including "blocks world" planning, integration problem solving, learning artifact descriptions, proving theorems in logic, and designing logic circuits. Nevertheless, the same learning system is used to generalize explanations for all of these examples. EGGS also includes performance systems that produce explanations for the learning system and in turn use the rules it generates to improve their ability to solve future problems.

EBG [Mitchell86] is an alternative domain-independent technique for generalizing explanations that was independently developed at the same time EGGS was originally designed and implemented. The eventual implementation of EBG [Kedar-Cabelli87a], has also been tested on several of the domains mentioned above. All of the original systems listed above used similar generalization techniques; however, until the development of EGGS and EBG, there was not a general learning technique that could be used to generalize examples in all of these domains. Explanation generalization is also a very computationally efficient process. As demonstrated in this thesis, explanation generalization can theoretically be performed in time linear in the size of the explanation.

EBL requires knowledge of the underlying theory of a domain before it can learn more useful knowledge for efficiently solving problems in that domain. Consequently, a general EBL mechanism that can learn in a variety of domains requires what has been called an *exchangeable knowledge module* [Michalski86]. In EGGS, this is accomplished by having both the generalizer and performance systems make use of a library of declarative knowledge that is uniformly represented. Currently, EGGS can rewrite rules, or STRIPS operators. This set of representational choices has allowed the efficient implementation of a numerous examples from various domains.

Finally, previous EBL systems that learn plans or macro-operators have generally been used to learn linear sequences of actions. EGGS is unique in that it contains a mechanism for learning plans for partially-ordered sequences of actions. This process results in more general plans which are applicable to a wider range of future situations.

1.2. GENESIS: Acquiring Schemata for Narrative Understanding

GENESIS is a narrative understanding system that improves its performance by using the EGGS learning system to acquire an abstract schema from a specific instance of a novel plan presented in an English narrative. As such, it tests the capacity of the EGGS learning system to improve the processing ability of a complete performance system operating in a complex domain.

Narrative text understanding was chosen as a performance domain for the following reasons. The ability to "understand" natural language text is a very difficult task requiring a large amount of world knowledge. Systems for understanding natural language text (e.g. [Cullingford78, DeJong82b, Dyer83]) generally encode relevant world knowledge in terms of *scripts* or *schemata* [Schank77]. The amount of world knowledge represented in terms of schemata largely determines the performance of such a system. Experience with the FRUMP system [DeJong82b] indicated that robustness of a text understanding system is directly related to the number of schemata it possesses. However, anticipating and encoding all of the schemata required for a robust natural language system is impossible for both theoretical and practical reasons. Theoretically, texts can display novel concepts unknown to the implementors of a natural language system. If the natural language system is to respond properly, it must discover such new concepts automatically. Practically, the number of schemata required to cover most natural language domains is prohibitively large and prevents manual programming of all of the necessary concepts. Once again, automatic schema acquisition is essential.

Of course, narrative understanding is not unique in this regard. Most realistic performance tasks require large amounts of domain knowledge represented in a form that can be efficiently used to solve problems; and constructing such a knowledge-base is known to be a difficult task. In fact, most work in machine learning is at least indirectly addressing the problem of the "knowledge acquisition bottleneck" [Feigenbaum83] in the construction of knowledge-based systems. However, narrative understanding has the advantage that issues can be illustrated with examples that are accessible to everyone, not just to individuals who have advanced degrees in medicine, geology, or computer engineering.

Another interesting aspect of this domain is that explanation-based learning has generally been used to improve the performance of a problem-solving system rather than an understanding system. GENESIS is unique in that it demonstrates that EBL can be used to improve a system's ability to explain observed behavior as well as its ability to solve problems efficiently.

1.3. Historical Comments

An original implementation of GENESIS was reported in [Mooney85b] (shorter versions include [Mooney85a] and [Mooney86a]). This version employed a relatively *ad hoc* explanation generalizing system that was specific to the representations used by the narrative understanding system (see [Mooney85b] for details on this generalizer). However, the original GENESIS generalizer was similar to the *lifting* process used to learn generalized plan MACROPS in STRIPS [Fikes72] and was also influenced by the TMS-based generalizer used by the MA system [O'Rorke87b].

An attempt to convert the original GENESIS generalizer into a domain-independent explanation generalizer lead to the development of the EGGS system. The generalization algorithm underlying the EGGS system was first described in [DeJong86b] and a comparison of EGGS, EBG, and STRIPS as well as examples of applying EGGS to numerous domains was first presented in [Mooney86c]. After the development of EGGS was complete, an entirely new version of the GENESIS system was built on top of this general explanation-based learning system. This version of GENESIS is described in detail in this thesis.[1]

1.4. Organization of the Thesis

The thesis begins with a sample performance of the GENESIS system that illustrates the sort of behavior exhibited by the complete system. The next four chapters discuss various components of the EGGS system. Chapter 3 discusses the domain-independent explanation generalizer that is used to generalize all of the examples presented in the thesis. It also includes a discussion on the relation between the explanation generalization algorithm used in EGGS and the generalizing algorithms used in STRIPS, EBG, and SOAR [Laird86b] as well as an analysis of the computational complexity of explanation generalization. Chapters 4-6 discuss additional sub-systems in EGGS for building explanations based on various representations and present numerous examples of using EGGS to learn in various domains. Chapter 7 presents a summary of empirical results on the effect of learning on future problem solving performance.

The next six chapters discuss additional components in the complete GENESIS system. Chapter 8 presents the architecture of the complete system and briefly discusses the role of each of its components. Chapter 9 discusses the knowledge representation used in GENESIS and Chapter 10 describes the processes used in understanding narratives. Chapter 11 discusses how GENESIS uses EGGS to generalize explanations constructed by the understanding system and illustrates how the schemata it learns improve its ability to understand subsequent narratives. The final two chapters about GENESIS discuss its ability to learn meanings for schema-related words from a single instance of their use (Chapter 12) and to use learned schemata to index and retrieve specific episodes (Chapter 13).

---

[1]For several reasons, this new implementation is not called GENESIS II. One reason is to avoid association with Gene Roddenberry's failed TV pilot of the same name [Gerrold73]. A second reason is to avoid the temptation to make as many versions of GENESIS as there were sequels to *Friday the 13th*. The latter behavior has been known to provoke satirical comments from some stone-throwing members of the machine learning community (e.g. the reference to AQ63 in [Gangly87]).

Chapter 14 reviews some psychological experiments that indicate explanation-based learning is psychologically plausible. Specifically, these experiments demonstrate that people, like GENESIS, can learn a schema from a single specific example presented in a narrative provided they have the knowledge to explain the actions in the example.

Chapter 15 summarizes the unique features of this research compared to other work in EBL and machine learning in general and discusses some problems and directions for future research.

Finally, there are a number of appendices that provide additional details. Appendix A presents a lemma needed for the complexity analysis of explanation generalization. Appendix B presents additional examples of GENESIS' performance which even the casual reader may find interesting to peruse. Appendix C lists all of GENESIS' initial knowledge and appendix D gives a detailed GENESIS trace.

1.5. Comments on Notation, Implementation, and Figures

Throughout the thesis, the LISP convention of using a leading question mark (e.g. ?x23) is used to denote predicate calculus variables [Charniak85]. In general, standard predicate calculus notation (e.g. P(?x,?y)) is preferred to LISP notation (e.g. (P ?x ?y)); however, for convenience, the latter is sometimes used, particularly in the appendices. All implementations are written in INTERLISP [Teitelman83] and were developed and run on a XEROX 1108 (Dandelion) with 3.5 MB of main memory. All explanation graphs presented in the thesis (including accompanying node labelling tables when needed) were generated automatically by the system using Interlisp-D's GRAPHER package [Xerox86] and automatically converted to TROFF PIC [Kernighan79] and TBL [Lesk79] descriptions. Only minor hand-editing was performed to reposition nodes for more compact lay-out and to insert special characters (e.g. $\int$, $\wedge$, $\cong$). When tables are used to fully label nodes in large explanation graphs, the order of the nodes follows a depth-first, left-to-right traversal of the graph.

CHAPTER 2

SAMPLE GENESIS PERFORMANCE

The purpose of this chapter is to provide the reader with an intuitive feel for the overall operation of the system. Throughout this thesis, it is assumed that the reader, like an EBL system, can learn a great deal by analyzing and understanding a single concrete example.

The primary goal of the GENESIS system is to improve its ability to understand natural language narratives by learning new plan schemata. Such schemata are acquired by explaining and generalizing a specific instance of a plan executed by a character in a narrative. Characters' actions are explained in terms of later actions that they enable and in terms of ultimate goals that they achieve. When the system detects that a character has achieved an important goal by combining actions in a novel and unfamiliar way, it generalizes the specific explanation for how the goal was achieved into a general plan schema. Generalization is performed by an analytic technique (EGGS) which removes irrelevant information while maintaining the validity of the explanation. The resulting schema is then retained by the system and indexed so that it can be subsequently retrieved and used to aid in the understanding of future narratives.

A standard procedure is used to test GENESIS' ability to learn a particular schema from a single instance. This procedure illustrates both the schema learning process itself as well as the ability of the learned schema to improve system performance. First, the system is given a *test narrative* which presents a sparse description of an instance of the schema. This description is missing one or more actions that are crucial to the overall plan. Consequently, the narrative is not detailed enough for the system to construct a causally complete explanation for characters' actions without a schema to supply missing actions and inferences. The system is therefore unable to answer questions that require making default inferences about what must have taken place and it is unable to produce an adequate paraphrase of the narrative. Next, the system is given a *learning narrative* which describes in detail a complete instance of the schema and which contains the crucial actions and other information that were lacking in the test narrative. Using its existing knowledge, GENESIS is able to construct a causally complete explanation for this narrative and can therefore answer questions about why actors performed certain actions as well as produce an adequate paraphrase of the text. Since the narrative presents a novel plan that achieves an important goal, the system generalizes its explanation of how the goal was achieved into a new plan schema. Finally, GENESIS is given the test narrative again. This time, it can use the schema it has just learned to fill in missing information, and as a result, it can now answer questions that it was previously unable to answer as well as produce a better paraphrase of the text.

The remainder of this chapter is dedicated to presenting the trace produced by GENESIS as it learns a "kidnapping for ransom" schema [DeJong81, DeJong82a, DeJong86a, DeJong86b, Mooney85a, Mooney86a]. This is currently the most complicated example upon which GENESIS has been tested. A complete step-by-step description of the system's performance on this example is not included in this thesis since it would necessarily be very long and complicated and the important processes are more easily illustrated using simpler examples. However, more detailed information about this example is given in appendix B and definitions of all of GENESIS' initial schemata are given in appendix C. Chapters 9-11 include a complete step-by-step description of how GENESIS learns and uses a "murder for inheritance" schema and a detailed system trace of this example is given in appendix D. Details on other GENESIS examples such as "arson for insurance" and "solicitation entrapment" are also given in appendix B.

Before processing the narratives used in the kidnapping example, the system has knowledge about bargaining, capturing and confining, threatening, and many other concepts; however, it does not have any knowledge about the concept of kidnapping for ransom nor any knowledge of the words "kidnap," "kidnapper," or "ransom." In order to illustrate the system's current ignorance of kidnapping and how this lack of knowledge affects its ability to understand certain kidnapping narratives, the system is first given a test narrative. Below is the actual I/O trace produced by the system.

Input: Ted is Alice's husband. He won 100000 dollars in the lottery. Bob imprisoned Alice in his basement. Bob got 75000 dollars and released Alice.

Thematic goal achieved: Ted is happy that Ted has the $100000.

Ready for questions:

>Summarize
Ted won $100000 in a lottery. Bob captured Alice and locked her in a basement. Bob released Alice.

>Why did Bob lock Alice in his basement?

Cannot find sufficient reason.

>How did Bob get the money?
Cannot find sufficient cause.

>Why did Bob release Alice
Cannot find sufficient reason.

>How did Ted get the money?
Ted won the $100000 in the lottery.

Notice that this narrative does not mention how Bob got the money or why Bob imprisoned Alice and then released her. Since the system does not have a schema for kidnapping, it cannot infer the missing information and causally connect the actions in the input. Consequently, its paraphrase of the narrative is very poor and it is unable to answer many questions about it. Next, GENESIS is given the following more detailed narrative:

Input: Fred is Mary's father and is a millionaire. John approached Mary and pointed a gun at her. She was wearing blue jeans. He told her if she did not get in his car then he would shoot her. He drove her to his hotel and locked her in his room. John called Fred and told him John was holding Mary captive. John told Fred if Fred gave him 250000 dollars at Trenos then John would release Mary. Fred paid him the ransom and the kidnapper released Mary. Valerie is Fred's wife and he told her that someone had kidnapped Mary.

Thematic goal achieved: John is happy that John has the $250000.
Explanation suitable for generalization. Pruning...Generalizing...Packaging...
Creating New Schema: (CaptureBargain ?x55 ?a34 ?b9 ?c4 ?r5 ?y5 ?l11)

?b9 is a person. ?c4 is a location. ?r5 is a room. ?c4 is in ?r5. ?x55 is a character. ?b9 is free. ?x55 captures ?b9 and locks him/her in ?r5. ?a34 is a character. ?x55 contacts ?a34 and tells it that ?b9 is ?x55's captive. ?y5 is a valuable. ?x55 wants to have ?y5 more than it wants ?b9 to be ?x55's captive. ?a34 has a positive relationship with ?b9. ?a34 has ?y5. ?x55 and ?a34 carry out a bargain in which ?x55 releases ?b9 and ?a34 gives ?x55 ?y5 at ?l11.

Unknown word 'ransom' refers to ?y5 in CaptureBargain
Unknown word 'kidnapper' refers to ?x55 in CaptureBargain

Having: ?x55 captured ?b9. suggest CaptureBargain
Having: ?x55 contacted ?a34 and told it that ?b9 was ?x55's captive. suggest CaptureBargain
Having: ?x55 and ?a34 carried out a bargain in which ?x55 released ?b9 and ?a34 gave ?x55 ?y5 at ?l11. suggest CaptureBargain

Unknown word '?x55 kidnap ?b9' refers to CaptureBargain

Ready for questions:

>Summarize
Mary was free. John captured Mary and locked her in a room. John contacted Fred and told him that Mary was John's captive. John wanted to have $250000 more than he wanted Mary to be John's captive. Fred had a positive relationship with Mary. Fred had the $250000. John and Fred carried out a bargain in which John released Mary and Fred gave John the $250000 at Trenos.

>Why did John approach Mary?
So John could aim the gun at Mary.

>Why did John aim the gun at Mary?
So John could threaten to shoot Mary with the gun unless Mary went from John into the car.

>Why did Mary get into the car?
Because Mary wanted not to be shot and because Mary believed that if Mary did not go from John into the car then John would shoot Mary with the gun.

>Why did John threaten to shoot Mary unless she got in the car?
So John could drive Mary to the hotel in the car.

>Why did John drive Mary to the hotel?

So John could move Mary from the hotel into the room.

>Why did John move Mary into the room?
So John could confine Mary in the room.

>Why did John confine Mary in the room?
So John could call Fred and could tell him that Mary was John's captive and so John and Fred could carry out a bargain in which John released Mary and Fred gave John the $250000 at Trenos.

>Why did John tell Fred that John was holding Mary captive?
So John and Fred could carry out a bargain in which John released Mary and Fred gave John the $250000 at Trenos.

>Why did Fred give John the money?
Because Fred believed that if Fred gave John the $250000 at Trenos then John would release Mary and because Fred wanted Mary to be free more than he wanted to have the $250000.

>Why did Fred have $250000?
Because Fred had millions of dollars.

>Why did Fred believe that John was holding Mary captive?
Because John called Fred and told him that Mary was John's captive.

>Why did Fred want Mary free?
Because Fred was Mary's father.

>Why did John release Mary?
Because John wanted to have the $250000 more than he wanted Mary to be John's captive and because John believed that if John released Mary then Fred would give John the $250000 at Trenos.

Unlike the first narrative, this one is detailed enough to allow GENESIS to causally connect the characters' actions. Specifically, the crucial bargain that took place between the kidnapper and the ransom payer is alluded to more directly. As a result, the system is able to answer numerous questions about why certain characters performed certain actions. The understanding mechanism is discussed in detail in chapter 10, and the specifics of understanding this particular narrative are given in appendix B. The resulting explanation for how John got the $250,000 is generalized into a new schema for kidnapping for ransom (which GENESIS calls CaptureBargain based on the names of two existing schemata that compose the new schema). Chapter 11 discusses the schema acquisition process in detail.

A few important aspects of the learned schema should be pointed out. First, it does not contain any facts or actions that are irrelevant to the workability of the plan. For example, the fact that the victim is wearing blue jeans is not included. A similarity-based learning system that was given a number of examples of kidnapping in which the victim was always wearing blue jeans is liable to include this fact in its representation of the concept. Second, the exact manner in which the component plan schemata (e.g. Capture) were decomposed into subgoals and executed in the example is not included in the schema. For example, the fact that the kidnapper executed the Capture by threatening the victim with a gun and then driving her to a hotel or that he contacted the ransom payer by telephone are not a part of the CaptureBargain schema. Third, the generalization process eliminates facts that are causally relevant to the plan only because they are specializations of more general facts. For example, the fact that the ransom payer was the victim's father is relevant to the plan since it motivated him to pay the ransom; however, the more important fact is the more general one that he had a positive emotional relationship with the victim and consequently valued her freedom more than personal possessions. This generalization is important since the schema should be able to handle alternative relationships between the ransom payer and victim such as the husband-wife relationship in the test narrative.

After learning CaptureBargain, the system indexes the new schema so that if it subsequently encounters a narrative in which someone captures someone or exchanges someone's freedom for a valuable item, it will be reminded of CaptureBargain and attempt to use it to help understand the narrative. Details of the indexing process and how learned schemata can aid the understanding of future narratives are also given in chapter 11. The system also acquires preliminary definitions for three unknown words which occurred in the story: kidnapper, ransom, and kidnap. GENESIS' ability to also learn word meanings from a single example is discussed in chapter 12.

Finally, GENESIS is given the test narrative once again. This time, Alice's imprisonment reminds it of CaptureBargain and it uses this newly acquired schema to infer the missing actions and explain how Bob got the money. Consequently, its ability to answer questions and paraphrase the text is greatly improved.

Input: Ted is Alice's husband. He won 100000 dollars in the lottery. Bob imprisoned Alice in his basement. Bob got 75000 dollars and released Alice.

Thematic goal achieved: Ted is happy that Ted has the $100000.
Thematic goal achieved: Bob is happy that Bob has the $75000.

Ready for questions:

>Summarize
Alice was free. Bob captured Alice and locked her in a basement. Bob contacted Ted and told him that Alice was Bob's captive. Bob wanted to have $75000 more than he wanted Alice to be Bob's captive. Ted had a positive relationship with Alice. Ted had the $75000. Bob and Ted carried out a bargain in which Bob released Alice and Ted gave Bob the $75000.

>Why did Bob lock Alice in his basement?
So Bob could contact Ted and could tell him that Alice was Bob's captive and so Bob and Ted could carry out a bargain in which Bob released Alice and Ted gave Bob the $75000.

>Why did Bob release Alice?
Because Bob wanted to have the $75000 more than he wanted Alice to be Bob's captive and because Bob believed that if Bob released Alice then Ted would give Bob the $75000.

>How did Bob get the money?
Bob kidnapped Alice.

>Who gave Bob the money?
Ted gave Bob the $75000.

>Why did Ted give him the money?

Because Ted believed that if Ted gave Bob the $75000 then Bob would release Alice and because Ted wanted Alice to be free more than he wanted to have the $75000.

>Why did Ted want Alice free?

Because Ted was Alice's husband.

>Why did Ted believe that Bob was holding Alice captive?

Because Bob contacted Ted and told him that Alice was Bob's captive.

While answering questions about this narrative for the second time, it is interesting to note that the system used the newly learned word "kidnap" to refer to the CaptureBargain schema in its answer to the question: "How did Bob get the money?" This is because the state in question is an effect of the new schema and since it knows how to refer to this schema in English, it considers it to be an appropriate answer.

CHAPTER 3

EGGS: A DOMAIN INDEPENDENT EXPLANATION GENERALIZER

EGGS is a general domain independent explanation-based learning system. In addition to being the underlying learning mechanism in the GENESIS system, it has been tested in numerous other domains using various underlying representations such as Horn-clause logic, rewrite rules, and STRIPS operators. An abstract outline of the learning process in EGGS is given in Figure 3.1 and an architectural diagram of the system is given in Figure 3.2.

The tasks of constructing an explanation (step 1) and packaging the generalized explanation for future use (step 4) both depend on the underlying representational formalism. Each representational formalism requires different modules for these tasks. For example, when using Horn clauses, a theorem prover is appropriate for constructing explanations, while when using STRIPS operators, a planner is appropriate. Each of the following three chapters is dedicated to a different representational formalism and discusses modules within EGGS for building and packaging explanations using that representation. Examples of using each representation in various domain are also given in these chapters.

Unlike explanation construction and packaging, explanation generalization (step 3), can be characterized in a very general way and is discussed in detail in this chapter. The general task of pruning the structure of the explanation to increase generality while maintaining operationality (step 2) is also characterized in this chapter; however, specific rules for pruning are domain dependent and are discussed in following chapters.

3.1. Explanations, Explanation Structures, and Generalized Explanations

In different domains, various types of explanations are appropriate. In [Mitchell86], an explanation is defined as a logical proof that demonstrates how an example meets a set of sufficient conditions defining a particular concept. This type of explanation is appropriate for learning classical concept definitions, such as learning a structural specification of a cup, an example introduced in [Winston83] and discussed in [Mitchell86]. However, when learning general plans in a problem solving domain (as in STRIPS [Fikes72] or GENESIS [Mooney85a]), it is more appropriate to consider an explanation to be a set of causally connected actions that demonstrate how a goal state is achieved.

1. Explain: Construct a complete plan or proof for a specific example by either doing independent problem solving to achieve a specified goal or by explaining the actions or operators executed by an external agent.

2. Prune: Remove branches of the explanation that are more specific than needed for the operationality of the resulting plan or proof.

3. Generalize: Generalize the remaining explanation as far as possible without invalidating its underlying structure.

4. Package: Create a macro-operator or macro-rule that summarizes the resulting generalized explanation and index it so that it can be used to aid future problem solving and understanding.

Figure 3.1: The Learning Process in EGGS

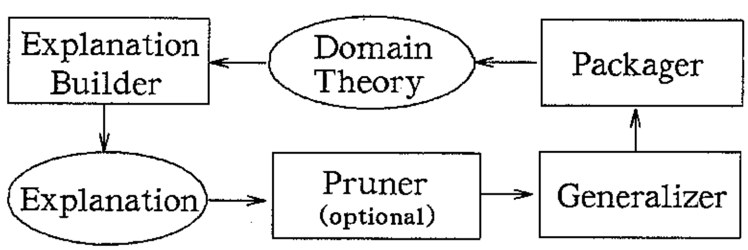


Figure 3.2: EGGS Architecture

Consequently, this work takes a very broad definition of the term *explanation* and considers it to be a connected set of *units*, where a unit is set of related well-formed-formulas (wffs) in predicate calculus. Horn-clause proofs, where each Horn clause is a unit, and plans composed of STRIPS operators, where each operator is a unit, are special cases of this very general representation. Formally, a unit can be defined as follows.

A **unit** is a connected directed acyclical graph (V, E) in which the vertices in V are wffs.

For example, a unit for a Horn-clause rule has wffs for its antecedents and its consequent, while a unit for a STRIPS operator has wffs for its effects, deletions, and preconditions. A wff a in a unit is said to *support* another wff b in the unit if and only if there is a directed path from a to b. For example, in the unit for a Horn clause, each antecedent *supports* the consequent through a path containing a single edge.

A *domain theory*, T, is formally defined as a set of units. As defined in [Nilsson80], a *substitution* is a set of ordered pairs each specifying a term to be substituted for a particular variable. The expression $p\theta$, where p is a wff and θ is a substitution, denotes the wff resulting from applying θ to p. The expression $\gamma\theta$, where both γ and θ are substitutions, denotes the substitution resulting from the *composition* of γ and θ, which is obtained by applying θ to the terms of γ and then adding any pairs of θ having variables not occurring among the variables of γ. An *instance* of a unit, α, is a unit obtained by applying a variable substitution to all of the wffs in α. Two wffs are said to be *identical* if and only if all of their corresponding predicates, functions, variables, and constants are exactly the same (i.e. their most general unifier is the null substitution). Before formally defining an explanation in this representation, a few additional definitions are needed.

A **unit-set** is a pair (U, R) where U is a set of units: $\{(V_1, E_1), \ldots (V_n, E_n)\}$ and R is an equivalence relation defined on the set of wffs: $V_1 \cup V_2 \ldots \cup V_n$. For each pair of wffs (a, b) in R where $a \in V_i$ and $b \in V_j$, it must be the case that that $i \neq j$ (i.e. equivalent wffs must be from separate units).

Given a unit-set S = (U, R) where $U = \{(V_1, E_1), \ldots (V_n, E_n)\}$, let $C_1, C_2, \ldots C_m$ be the equivalence classes of wffs defined by R. Let G be the graph (V′, E′) where $V' = \{C_1, C_2, \ldots C_m\}$ and $(C_i, C_j) \in E'$ if and only if there are wffs $a \in C_i$ and $b \in C_j$ such that $(a, b) \in E_1 \cup E_2 \ldots \cup E_n$. G is referred to as *the graph of* S and represents the directed graph obtained by "collapsing" all equivalent wffs into a single vertex. A formal definition of an explanation can now be stated as follows.

An **explanation** is a unit-set, S = (U, R), where the graph of S is connected and acyclic and where for each pair of wffs $(a, b) \in R$, a and b are identical. Furthermore, let the set $U' \subset U$ be the set of all units in U that are instances of units in the domain theory, T, and let R′ be the equivalence relation such that $(a, b) \in R'$ if and only if both a and b are wffs from units in U′ and $(a, b) \in R$. In order for S to be an explanation, U′ must be nonempty and the graph of the unit-set S′ = (U′, R′) must also be connected and acyclic.

In other words, an explanation is a combination of units that forms an even larger connected acyclic graph by means of an an equivalence relation defined on their vertices. Each pair of wffs that the relation defines as equivalent must be identical. Furthermore, if all units that are not instances of units in the domain theory are removed from an explanation, the remaining explanation also defines a connected directed acyclic graph. The *goal* is a distinguished wff in the explanation that is a sink of the graph of the explanation and represents the final conclusion in an inference chain or the desired state in a plan.

A Horn-clause proof in this representation is an explanation whose units are Horn clauses and whose equivalence relation matches antecedents of some clauses to consequents of others. In this case, an explanation is analogous to the *data dependency* structure maintained by a *truth maintenance system* [Doyle79]. For example, consider the following domain theory for the problem of learning a structural definition of a cup, an example originally presented in [Winston83].

Stable(?x) $\wedge$ Liftable(?x) $\wedge$ OpenVessel(?x) $\rightarrow$ Cup(?x)

Bottom(?y) $\wedge$ PartOf(?y,?x) $\wedge$ Flat(?y) $\rightarrow$ Stable(?x)

Graspable(?x) $\wedge$ Light(?x) $\rightarrow$ Liftable(?x)

Handle(?y) $\wedge$ PartOf(?y,?x) $\rightarrow$ Graspable(?x)

Concavity(?y) $\wedge$ PartOf(?y,?x) $\wedge$ UpwardPointing(?y) $\rightarrow$ OpenVessel(?x)

Additional units needed for the problem are the following individual facts.

Light(Obj1), Color(Obj1,Red), PartOf(Handle1,Obj1), Handle(Handle1), Bottom(B1), PartOf(B1,Obj1), Flat(B1), Concavity(C1), PartOf(C1,Obj1), UpwardPointing(C1)

A proof tree for Cup(Obj1) is shown in Figure 3.3 as an explanation whose goal is Cup(Obj1). Triple edges in the graphs indicate equivalences between wffs in two units that are instances of the domain theory while double edges indicate equivalences to wffs in units that are not instances of units in the domain theory. Specifically, for explanations using Horn clauses as units, triple edges indicate connections between instantiations of rules from the domain theory while double edges indicate connections to initial facts about the specific example. Examples of explanations where the units include STRIPS operators are presented in chapter 6.

A wff a is a *uniquized version* of a wff b if and only if a is obtained by substituting a uniquely named variable for each variable in b. In correspondence with the terminology in [Mitchell86], an *explanation structure* is defined as an explanation with each instantiated unit from the domain theory replaced by a uniquized version of its general definition. Formally:

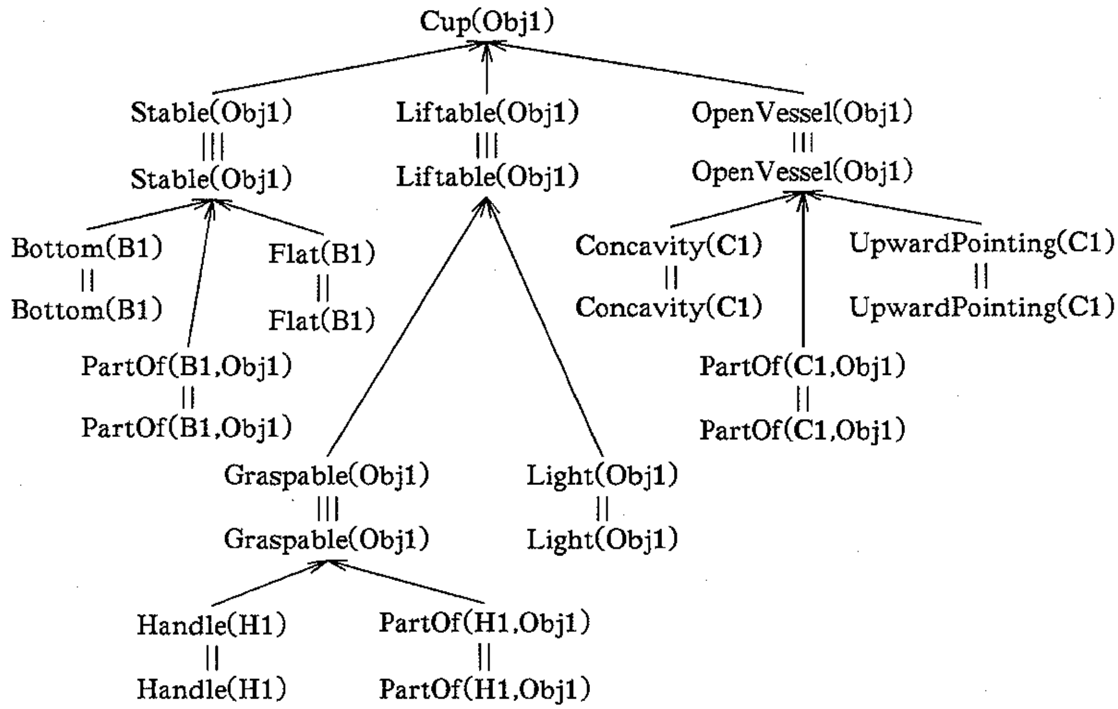


Figure 3.3: Explanation for Cup(Obj1)

An **explanation structure** of an explanation E = (U, R) is a unit-set, S = (U′, R′) where for each $u_i \in U$ where u_i is an instance of a unit $t_i \in T$, there is exactly one $u_i' \in U'$ such that u_i' is a uniquized version of t_i and where $(u_i', u_j') \in R'$ if and only if $(u_i, u_j) \in R$.

The definition of an explanation insures that an explanation structure defines a connected directed acyclic graph. For example, the explanation structure of the explanation for the cup example is shown in Figure 3.4.

The task of *explanation generalization* is to take an explanation and generate a *generalized explanation*, which is the most general instance of its explanation structure in which equivalent wffs are identical. The generalized explanation maintains matches between wffs from rules or facts in the domain theory but eliminates matches to wffs specifying facts of the particular specific example. This means that the most general substitution that results in all equivalent wffs being identical must be applied to the explanation structure. Formally:

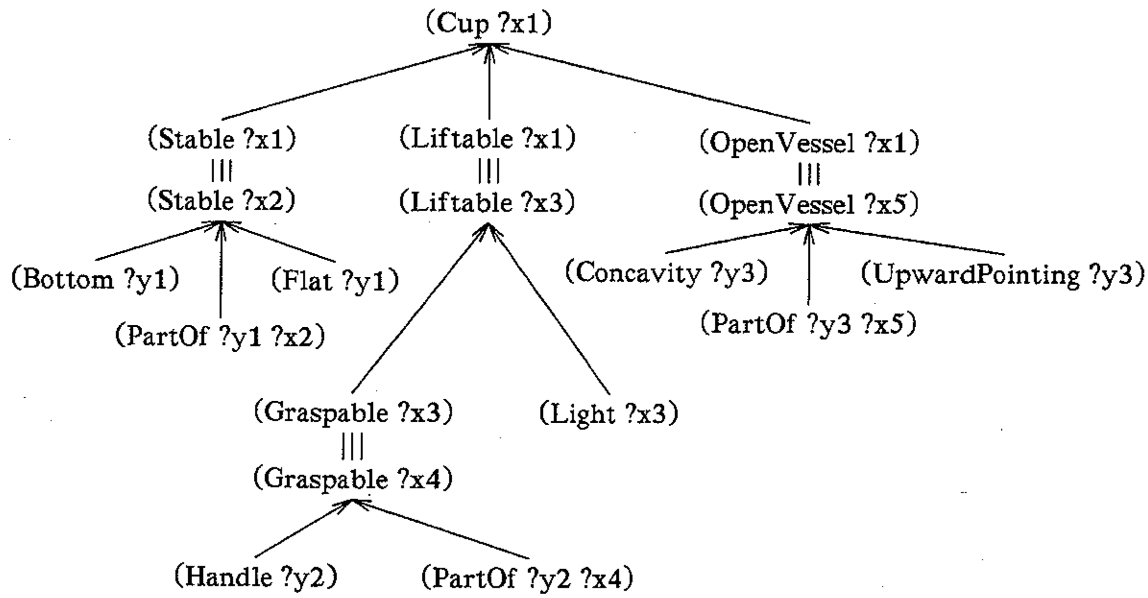


Figure 3.4: Explanation Structure for the Cup Example

A **generalized explanation** of an explanation E with an explanation structure S = (U, R) is an explanation G = (U′, R′) such that there exists a substitution γ where for each $u_i \in U$ there is exactly one $u_i' \in U'$ such that $u_i' = u_i\gamma$ and where $(u_i', u_j') \in R'$ if and only if $(u_i, u_j) \in R$. Furthermore, if $(u_i', u_j') \in R'$, then u_i' and u_j' must be identical. Finally, for any other substitution, θ, satisfying these constraints, there must exist a substitution θ' such that $U\theta = U\gamma\theta'$ (this insures that γ is the *most general* substitution that satisfies the constraints).

The generalized explanation of the cup example is shown in Figure 3.6. This generalized explanation can then be used to obtain the following *macro-rule* representing a general structural definition of a cup.

$$\text{Bottom(?y1)} \wedge \text{PartOf(?y1,?x1)} \wedge \text{Flat(?y1)} \wedge \text{Handle(?y2)} \wedge \text{PartOf(?y2,?x1)}$$
$$\wedge\ \text{Light(?x1)} \wedge \text{Concavity(?y3)} \wedge \text{PartOf(?y3,?x1)} \wedge \text{UpwardPointing(?y3)} \rightarrow \text{Cup(?x1)}$$

For explanations that are logical proofs, a macro-rule like the one above is easily obtained by taking the leaves of the generalized explanation as the antecedents and the goal of the generalized explanation as the consequent. In planning domains, the generalized explanation represents a

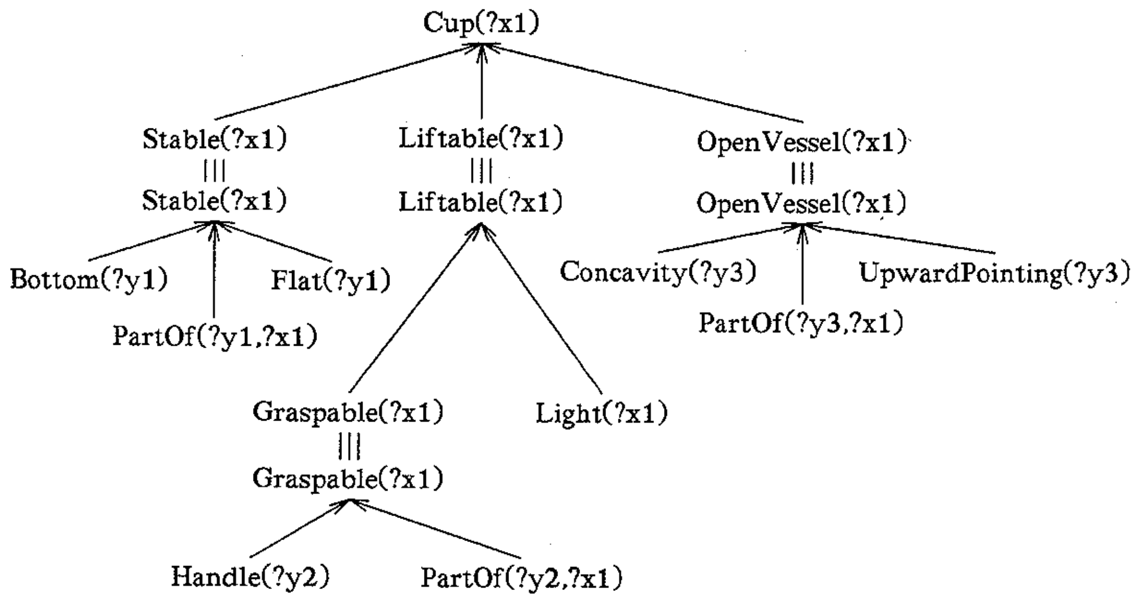


Figure 3.5: Generalized Explanation for the Cup Example

general plan schema or macro-operator [Fikes72] for achieving a particular class of goals. Creating a new action definition for the composed plan from the generalized explanation requires a few additional steps which are discussed in section 6.2.2.

3.2. Explanation Generalizing Algorithms

Several algorithms have been developed for generalizing various types of explanations. The STRIPS system [Fikes72] incorporated a method for generalizing blocks-world plans into macro-operators. The EBG method [Mitchell86] uses a modified version of goal-regression [Waldinger77] to generalize proofs of concept membership. The EGGS explanation generalization algorithm was developed for generalizing the broad class of explanations defined in the previous section. This algorithm was first published in [DeJong86b] along with a description of an error found in the specification of the EBG algorithm. Kedar-Cabelli and McCarty subsequently developed a PROLOG version of EBG [Kedar-Cabelli87a] which corrected this problem with the original algorithm.

The general technique used by STRIPS, EBG, EGGS, and PROLOG-EBG can be abstracted to apply to the class of explanations defined in the previous section. The rest of this section is devoted to presenting and comparing algorithmic descriptions of all of these methods as applied to this class of explanations. All of the algorithms rely on unification pattern matching and the abbreviation MGU is used to refer to the substitution that is the *most general unifier* of two wffs [Charniak80, Nilsson80]. All of the generalization algorithms presented have been implemented and tested within the context of the overall EGGS system.

3.2.1. STRIPS MACROP Learning

The first work on generalizing explanations was the learning of robot plans in STRIPS [Fikes72]. STRIPS worked in a "blocks world" domain. After its problem solving component generated a plan for achieving a particular state, it generalized the plan into a problem solving schema (a MACROP or macro-operator) which could be used to efficiently solve similar problems in the future. Work on the STRIPS system was the first to point out that a correct generalization of a connected set of actions or inferences can *not* be obtained by simply replacing each constant by an independent variable. This method happens to work on the CUP example given above. The proper generalized explanation can be obtained by replacing Obj1 by ?x1, B1 by ?y1, H1 by ?y2, and C1 by ?y3. However, in general, such a simplistic approach can result in a structure that is either more general or more specific than what is actually supported by the system's domain knowledge.

The following examples are given in [Fikes72] to illustrate that simply replacing constants with variables can result in improper generalizations. These examples assume the initial state shown in Figure 3.6 and use the following operators:

GoThru(?d,?r1,?r2):	Go through door ?d from room ?r1 to room ?r2.
PushThru(?b,?d,?r1,?r2):	Push box ?b through door ?d from room ?r1 to room ?r2.
SpecialPush(?b):	Specific operator for pushing box ?b from Room2 to Room1.

Given the plan:

GoThru(Door1,Room1,Room2)
SpecialPush(Box1)

simply replacing constants by variables results in the plan:

GoThru(?d,?r1,?r2)
SpecialPush(?b)

This plan is too general since SpecialPush is only applicable when starting in Room2, so having a variable ?r2 as the destination of the GoThru is too general and ?r2 should be replaced by Room2.

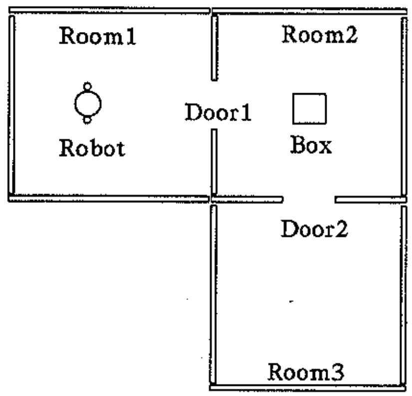


Figure 3.6: Initial World State for STRIPS Examples

Given the plan:

GoThru(Door1,Room1,Room2)

PushThru(Box1,Door1,Room2,Room1)

simply replacing constants by variables results in the plan:

GoThru(?d,?r1,?r2)

PushThru(?b,?d,?r2,?r1)

This plan is too specific since the operators themselves do not demand that the room in which the robot begins (?r1) be the same room into which the box is pushed. The correct generalization is:

GoThru(?d,?r1,?r2)

PushThru(?b,?d,?r2,?r3)

The exact process STRIPS uses to avoid these problems and correctly generalize an example is dependent on its particular representations (triangle tables) and inference techniques (resolution); however, the basic technique is easily captured using the representations discussed in section 3.1. A description of the basic explanation generalizing algorithm used in STRIPS is shown in Figure 3.7. It should be noted that the generalization process in STRIPS was constructed specifically for generalizing robot plans. There was no attempt to present a general learning method based on generalizing explanations in any domain. However, the algorithm in Figure 3.7 is a straight-forward generalization of the basic process used in STRIPS. The basic technique is to unify each pair of

for each equality between wffs x and y in the explanation structure **do**
  let θ be the MGU of x and y
  for each wff z in the explanation structure **do**
    replace z **with** $z\theta$

Figure 3.7: STRIPS Explanation Generalizing Algorithm

equivalent wffs in the explanation structure and apply each resulting substitution to all of the wffs in the explanation structure. After all of the unifications and substitutions have been made, the result is the generalized explanation since each wff has been replaced by the most general wff that allows all of the equality matches in the explanation to be satisfied.

3.2.2. EBG

In [Mitchell86], Mitchell, Keller, and Kedar-Cabelli describe a technique called EBG (Explanation-Based Generalization) for generalizing a logical proof that a particular example satisfies the definition of a concept. An example concept-membership proof showing how a particular object satisfies the functional definition of a cup was given in Figure 3.3. Unlike the STRIPS MACROP learning method, EBG was meant to be a general method for learning by generalizing explanations of why an example is a member of a concept. In [Mitchell86], detailed examples are presented illustrating how EBG can be applied to learning an operational definition for when it is safe to stack something on an endtable[1], to Winston's CUP example [Winston83], and to an example from LEX2's domain of learning heuristics for symbolic integration [Mitchell83]. A much more abstract description of how it might be used to learn a kidnapping plan like that learned by the original GENESIS system [Mooney85a] is presented in an appendix.

The original EBG algorithm presented in [Mitchell86] is based on *goal regression* [Waldinger77] and involves back-propagating constraints from the goal through the explanation structure to the leaves. Figure 3.8 presents a formal specification of the original algorithm in terms of the explanation representation introduced earlier. The global variable R maintains the current set of regressed expressions and represents the most general set of antecedents necessary to prove the goal given the portion of the explanation structure already traversed. The explanation structure is traversed from the goal back to the leaves in a depth first manner. Each time a unit (rule) is traversed, the set R is updated and the substitution resulting from the unit's unification to the

```
let g be the goal wff in the explanation structure
let R be the set of wffs supporting g
EBG(g)

procedure EBG(p)
   for each wff x supporting p do
      if x is equivalent to some wff e
         then
            let R = R − {x}
            for each wff y supporting e do
               let R = R ∪ {y}
            let θ be the MGU of e and x
            for y in R do
               replace y with yθ
            EBG(e)
```

Figure 3.8: Original EBG Explanation Generalizing Algorithm

structure already traversed is applied to all of the wffs in R. After the entire explanation structure has been traversed, R is the most general set of antecedents for the given explanation structure.[1]

However, as initially pointed out in [DeJong86b], this algorithm is only guaranteed to determine the leaves of the generalized explanation and in certain situations fails to obtain the correct generalized goal. The "Suicide" example, originally introduced in [DeJong86b] is an example for which the original EBG algorithm does not compute the correct generalized goal and as a result learns an incorrect macro-rule. This example involves inferring that an individual will commit suicide if he is depressed and buys a gun. The specific facts of the problem are:

Depressed(John), Buy(John,Obj1), Isa(Obj1,Gun)

The domain rules are:

Depressed(?x) → Hate(?x,?x)

Hate(?x,?y) ∧ Possess(?x,?z) ∧ Isa(?z,Weapon) → Kill(?x,?y)

Buy(?x,?y) → Possess(?x,?y), Isa(?x,Gun) → Isa(?x,Weapon)

[1]The algorithm presented in Figure 3.8 corrects problems with the BackPropagate function presented in [Mooney86b, Mooney86c]. As discussed in [Mooney86b], the BackPropagate function (which was based on the informal description of this process given in [Mitchell86]) does not properly propagate constraints across conjuncts and consequently in some situations does not compute the correct regressed expressions. The version in Figure 3.8 does not have this problem since each substitution is applied to all of the current regressed expressions in the set R.

The proof that John will commit suicide is shown in Figure 3.9, its explanation structure is shown in Figure 3.10, and the correct general proof that anyone who is depressed and buys a gun will commit suicide is shown in Figure 3.11. The general macro-rule learned from the generalized explanation is:

$$\text{Depressed}(?y1) \wedge \text{Buy}(?y1,?c1) \wedge \text{Isa}(?c1,\text{Gun}) \rightarrow \text{Kill}(?y1,?y1)$$

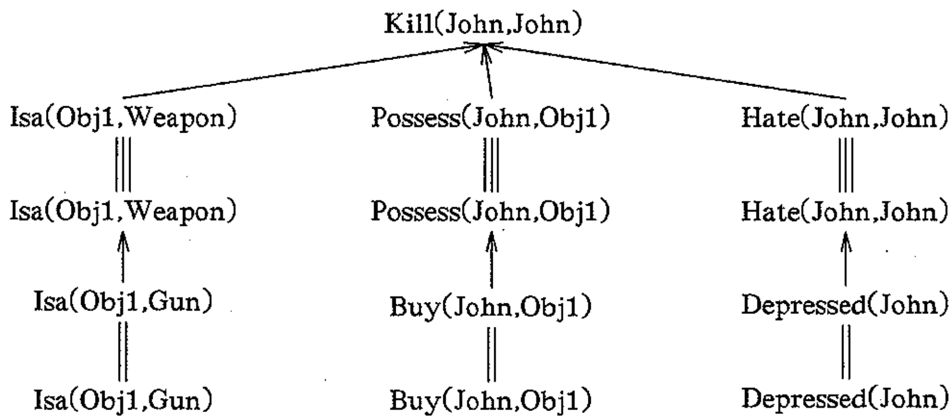


Figure 3.9: Suicide Example -- Specific Explanation

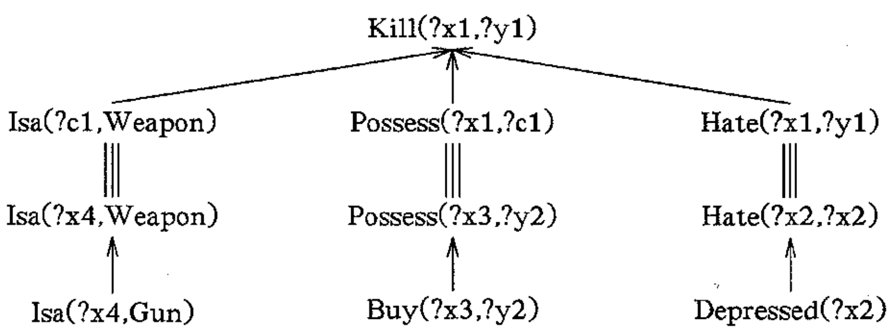


Figure 3.10: Suicide Example — Explanation Structure

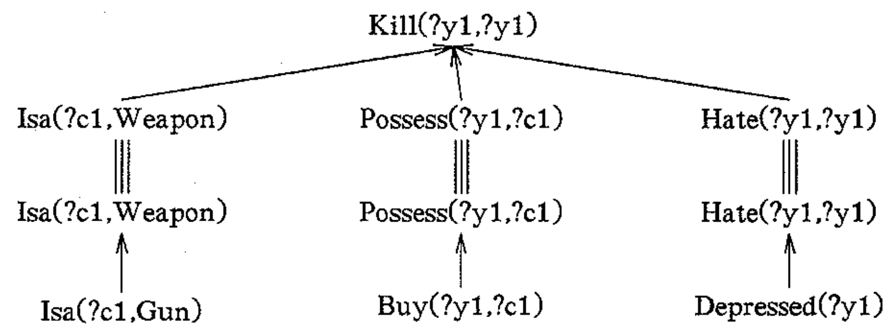


Figure 3.11: Suicide Example – Generalized Explanation

Goal regression, as given in [Mitchell86] and Figure 3.11, computes only the most general set of antecedents that would support a proof with the same explanation structure as the training example (i.e. the weakest preconditions [Dijkstra76, Minton84].). If only goal regression is performed, the proper description of the goal concept supported by the explanation is not always determined since the explanation itself may impose constraints on the goal concept. In terms of the Suicide example, the constraint that the killer be the same as the person killed is never imposed and, as demonstrated in [DeJong86b], EBG constructs the following erroneous rule:

Depressed(?y) ∧ Buy(?y,?c) ∧ Isa(?c,Gun) → Kill(?x,?y)

This rule states that everyone kills someone who is depressed and buys a gun, which is clearly not a conclusion warranted by the domain theory. Since the abstract STRIPS algorithm applies substitutions generated by each unification to the entire explanation structure, it computes the appropriately constrained goal concept and does not make this mistake.

As suggested in [DeJong86b], the proper generalized goal and generalized explanation can be obtained by starting with the generalized antecedents obtained from regression and rederiving the general proof. Rederiving the proof propagates constraints from the regressed expressions to the goal, thereby appropriately constraining the goal concept. The resulting generalization algorithm is then a two step process: goal regression (back-propagation) followed by proof reconstruction (forward propagation). This approach was suggested based on a similar two-pass generalization process presented in [Mahadevan85]. A formal description of a version of EBG corrected in this manner is given in [Mooney86b].[2] In [Kedar-Cabelli87a], Kedar-Cabelli and McCarty presented a PROLOG version of EBG which also corrects the problem and avoids making two separate passes through the explanation. This version of EBG is considered in a subsequent section.

3.2.3. EGGS

The EGGS (Explanation Generalization using a Global Substitution) algorithm was developed for generalizing explanations of the abstract form defined and used in this paper. The algorithm is quite similar to the abstract STRIPS algorithm and is shown in Figure 3.12.[3] The difference between EGGS and the abstract STRIPS algorithm is that instead of applying the substitutions throughout the explanation at each step, all the substitutions are composed into one substitution γ. After all the unifications have been performed, one sweep through the explanation applying the accumulated substitution γ results in the generalized explanation. Table 3.1 demonstrates this technique as applied to the Cup example. It shows how γ changes as it is composed with the substitutions resulting from each unification. Applying the final substitution γ to the explanation structure shown in Figure 3.4 results in the generalized explanation shown in Figure 3.5. Table 3.2 shows how EGGS generalizes the Suicide example. Applying the final substitution to the explanation structure shown in Figure 3.10 results in the generalized explanation shown in Figure 3.11. In the tables, equalities are processed in the order produced by depth-first traversals of the explanation

let γ be the null substitution {}
for each equality between patterns x and y in the explanation structure **do**
    let θ be the MGU of xγ and yγ
    let γ be $\gamma\theta$
for each wff x in the explanation structure **do**
    replace x **with** xγ

Figure 3.12: EGGS Explanation Generalizing Algorithm

[2] Since, as mentioned in the previous footnote, the BackPropagate function presented in [Mooney86b] does not properly propagate constraints across conjuncts, the corrected version of EBG presented in [Mooney86b] required that ForwardPropagate be performed before BackPropagate. If the correct version of back-propagation presented in Figure 3.8 is used, it is not necessary to perform forward propagation first.

[3] Due to a typographical error, the EGGS algorithm presented in [Mooney86c] did not include applying γ to x and y prior to computing their MGU. The original publication of the algorithm in [DeJong86b] did not suffer from this mistake and the longer technical report version of [Mooney86c] included a corrected version of the EGGS algorithm [Mooney86b].

Table 3.1: EGGS Applied To the Cup Example		
Equality	θ	γ
Stable(?x1) ≡ Stable(?x2)	{?x1/?x2}	{?x1/?x2}
Liftable(?x1) ≡ Liftable(?x3)	{?x1/?x3}	{?x1/?x2, ?x1/?x3}
Graspable(?x3) ≡ Graspable(?x4)	{?x1/?x4}	{?x1/?x2, ?x1/?x3, ?x1/?x4}
OpenVessel(?x1) ≡ OpenVessel(?x5)	{?x1/?x5}	{?x1/?x2, ?x1/?x3, ?x1/?x4, ?x1/?x5}

Table 3.2: EGGS Applied To the Suicide Example		
Equality	θ	γ
Isa(?c1,Weapon) ≡ Isa(?x4,Weapon)	{?c1/?x4}	{?c1/?x4}
Possess(?x1,?c1) ≡ Possess(?x3,?y2)	{?x1/?x3, ?c1/?y2}	{?c1/?x4, ?x1/?x3, ?c1/?y2}
Hate(?x1,?y1) ≡ Hate(?x2,?x2)	{?y1/?x2, ?y1/?x1}	{?c1/?x4, ?x1/?x3, ?c1/?y2, ?y1/?x2, ?y1/?x1}

structures; however, any order will result in equivalent generalized explanations up to a change of variable names.

3.2.4. PROLOG-EBG

In [Kedar-Cabelli87a], Kedar-Cabelli and McCarty present a PROLOG version of EBG which, unlike the original EBG, computes the proper goal concept. PROLOG-EBG integrates the generalization process with the construction of explanations by PROLOG. A generalized proof is constructed in parallel with the proof for the specific example. Any query results in both a specific and and a generalized proof being returned.

The algorithmic description presented in Figure 3.13 is an attempt to specify the generalization algorithm underlying PROLOG-EBG as an independent process (i.e. separated from the process of theorem proving). Like EGGS, PROLOG-EBG constructs a global substitution, γ, which is then applied to the complete explanation structure. However, unlike EGGS, γ is constructed by traversing the explanation depth-first from the goal in a manner analogous to trying to prove the general goal of the explanation structure using backward-chaining. The substitution γ is constructed by finding a substitution that allows the goal to be proved from the set of operational wffs represented by the leaves of the explanation structure. The generalization algorithm is analogous to the algorithm for a backward-chaining deductive system (like PROLOG or the deductive retrieval system in [Charniak80]). In the algorithm in Figure 3.13, the function PROLOG-EBG returns two values: the current substitution (γ) and the subset of the wffs in the explanation structure that have already been traversed (E).[4] When the top-level call to PROLOG-EBG returns, E is the set of all wffs in the explanation structure and γ is the final global substitution.

It should be noted that Hirsh [Hirsh87] simultaneously developed a version of EBG for logic programming (using the MRS logic programming system [Russell85]) which uses a generalization algorithm that is equivalent in operation to PROLOG-EBG's. In addition to integrating theorem proving and generalization, MRS-EBG integrates both of these with operationality checking [Mitchell86] or *pruning* [DeJong86b, Mooney86c], a process discussed in section 3.6.

3.3. Comparison of Explanation Generalizing Algorithms

It is reasonably clear that STRIPS, EGGS, and EBG-PROLOG all compute the same desired generalized explanation. They all perform a set of unifications and substitutions that constrain the

---

```
let g be the goal wff in the explanation structure
let (γ, E) = PROLOG-EBG(g, {})
for e in E do replace e with eγ

procedure PROLOG-EBG(x, θ)
   let S be the set of wffs supporting x
   for s in S do replace s with sθ
   let (γ, E) = PROLOG-EBG-Supporters(S, {}, ∅)
   return (γθ, E ∪ {x})

procedure PROLOG-EBG-Supporters(S, γ, E)
   if S=∅
      then return (γ, E)
      else
         let f be the first element of S
         let R = S − {f}
         if f is equivalent to some wff e
            then
               let φ be the MGU of f and e
               let (δ, P) = PROLOG-EBG(e, φ)
               for r in R do replace r with rδ
               PROLOG-EBG-Supporters(R, γδ, E ∪ P)
            else PROLOG-EBG-Supporters(R, γ, E ∪ {f})
```

Figure 3.13: PROLOG-EBG Explanation Generalizing Algorithm

---

[4]The notation "(a, b)= F(x)" and "return (a, b)" is used to denote the fact that the function F returns two values: a and b. All variables referenced by a procedure are assumed to be local to that procedure call.

explanation structure into one in which equivalent wffs are identical. The difference between them lies in the manner and order in which the unifications and substitutions are done. As described in [O'Rorke87b], explanation-based generalization can be viewed as a process of posting and propagating equality or co-reference constraints. Neither the STRIPS or EGGS algorithm impose an ordering on the assimilation of the various equality constraints in the explanation structure. On the other hand, the various EBG algorithms order the assimilation of constraints by traversing the explanation structure depth-first. Although this ordering is not required by the generalization process, it is a natural consequence of integrating generalization with a backward-chaining theorem prover.

Actually, the task of producing a global substitution (γ) for an explanation structure can be easily shown to reduce to the task of finding a most general unifier for two wffs. The two wffs for the reduction are constructed by having equivalent wffs in the explanation structure occupy corresponding argument positions in the two constructed wffs. For example, below are two wffs constructed for the explanation structure of the CUP example (Figure 3.4).

P(Stable(?x2), Liftable(?x3), Graspable(?x4), OpenVessel(?x5))

P(Stable(?x1), Liftable(?x1), Graspable(?x3), OpenVessel(?x1))

A MGU for these two wffs is: {?x1/?x2, ?x1/?x3, ?x1/?x4, ?x1/?x5} which is the same as the global substitution EGGS constructed for this example (Table 3.1). The two wffs constructed for the Suicide example are:

P(Isa(?x4,Weapon), Possess(?x3,?y2), Hate(?x2,?x2))

P(Isa(?c1,Weapon), Possess(?x1,?c1), Hate(?x1,?y1))

A MGU for these two wffs is: {?c1/?x4, ?x1/?x3, ?c1/?y2, ?y1/?x2, ?y1/?x1} which is again the same as the global substitution constructed by EGGS (Table 3.2)

Consequently, in some sense the various explanation generalizing algorithms are just different ways of implementing unification. In fact, EGGS directly corresponds to the implementation of UNIFY in [Charniak80] which takes a pair of wffs and a current substitution and returns an updated substitution which includes variable bindings that unify the two wffs in the context of the current substitution. The STRIPS generalizing algorithm, on the other hand, is more similar to the implementation of UNIFY presented in [Nilsson80] in which the substitution unifying the first elements of two wffs is applied to the rest of the wffs before continuing. A unification algorithm that applied each substitution to the entire wff (thereby generating the resulting unified wff as well as a unifying substitution) would be exactly equivalent to the STRIPS generalizing algorithm.

3.4. Computational Complexity of Explanation Generalization

The reduction to unification given above also demonstrates that the time complexity of producing a global substitution is linear in the size of the explanation since linear time algorithms exist for unification [Paterson78]. Since unification can be performed in linear time, obviously the size of the resulting MGU must also be linear in the size of the explanation since only a linear amount of output can be produced in linear time. Therefore, if $|E|$ represents the size of the explanation, we can let $c_1|E|$ be the time required to construct the global substitution, and $c_2|E|$ be the length of the global substitution. Since the time complexity of applying a substitution to a wff is also linear in the length of its inputs[5], let $c_3(c_2|E| + |E|)$ be the time required to apply the global substitution to the explanation. Therefore, the time required for the complete process of constructing a generalized explanation is:

$$c_3(c_2|E| + |E|) + c_1|E| = (c_1 + c_3(c_2 + 1))|E|$$

which is clearly linear in size of the explanation.

Although this result does not reveal the time complexity of the individual algorithms in section 3.2, it is a constructive proof of the existence of a linear-time explanation generalizing algorithm. Since linear-time unification algorithms have apparently found limited use in practice due to large overhead, it is unlikely that a generalizing algorithm based on one would be particularly useful in practice. Nevertheless, it is an interesting theoretical result which supports the important claim that a generalized explanation can be computed very efficiently.

In practice, the generalizing algorithms given in section 3.2 are quite efficient using a standard non-linear unifier. In chapter 7, some empirical results on EGGS are presented which include data on the time required to learn from various examples. However, the learning times presented there include the time required for pruning the explanation and packaging it into a macro-rule as well as the time required for generalization. Of the problems reviewed in chapter 7, the most complicated one to generalize is the "difficult geometry" problem which has 25 equalities and takes only 3.6 seconds of CPU time to generalize.

3.5. Correctness of Explanation Generalizing Algorithms

Intuitively, in order for an explanation generalizing algorithm to be "correct," its output should be logically entailed by the system's existing knowledge or domain theory and it should be

[5]The literature on linear unification does not discuss linear time substitution application; however, a linear time algorithm for this procedure is presented in appendix A.

as general as possible given this constraint and the constraint that it retain the "structure" of the original explanation. To date there are no complete formal proofs of the correctness of any of the above generalization algorithms; and no such proofs will be attempted in this section. However, proving correctness does not seem to be an impossible task and this section summarizes several approaches that could be taken.

If it is assumed that explanations are logical proofs (as in [Mitchell86]), one would first want to prove *soundness*, i.e. that the learned macro-rule is logically entailed by the existing domain theory. For PROLOG-EBG, it is particularly clear that the generalization algorithm is simply performing deduction on Horn clauses and thereby constructing a proof of the generalized goal concept from the operational leaves of the explanation. Consequently, proving soundness could make productive use of existing proofs of the correctness of the deductive algorithms underlying PROLOG [Lloyd84, Robinson65]. However, proving that the learned macro-rule is the most general one possible would probably be more difficult.

If generalization is done by performing a single unification as described in section 3.3 (and the unification algorithm has been proven correct), proving soundness for logical proof explanations becomes straight-forward. In this approach, all of the equivalent wffs in the generalized explanation must be identical since unification can be proven to produce a substitution that will make its arguments identical [Robinson65]. Since all of the Horn clauses in the generalized explanation are instantiations of clauses in the domain theory (i.e. they are the result of applying the global substitution to the explanation structure), the logically sound inference rule of *universal instantiation* guarantees that they are entailed by the domain theory. Finally, one needs to show that computing a macro-rule is simply performing logically sound deduction on the Horn clauses in the generalized explanation. If

$$k_1 \dots k_{i-1} \wedge k_i \wedge k_{i+1} \dots k_n \rightarrow c$$

and

$$l_1 \wedge \dots \wedge l_n \rightarrow d$$

are two clauses in the generalized explanation, and d is equivalent to k_i, then d and k_i must be identical literals. Assume the second clause is removed from the generalized explanation and the first clause is replaced by:

$$k_1 \dots k_{i-1} \wedge l_1 \wedge \dots \wedge l_n \wedge k_{i+1} \dots k_n \rightarrow c$$

Since d and k_i are identical literals, the added clause is entailed by the domain theory because it is the resolvent of the two clauses and the resolution rule is sound [Robinson65]. Repeating this process for every set of equivalent patterns reduces the generalized explanation to the desired macro-rule. Since all of the clauses in the original generalized explanation are entailed by the domain theory, and since the clause added by each step in the reduction process is entailed by the existing clauses, by induction, the completely reduced generalized explanation (i.e. the learned macro-rule) is entailed by the domain theory.

If it could be proven that a generalization algorithm such as EGGS or EBG computes the correct global substitution and is therefore equivalent to the single unification algorithm, then it would follow that this algorithm was also sound. For EGGS, this would involve proving that the unification algorithm in [Charniak80] is correct.

Another approach to proving correctness of explanation generalization is discussed in [O'Rorke87b] and involves demonstrating that a generalization algorithm maintains the equality or co-reference constraints of the explanation structure in the most general way possible. As described in [O'Rorke87b], explanation generalization can be performed by combining the individual co-reference constraints in order to compute the most general description of each expression in the explanation that satisfies all of these constraints. More details on this approach to verification and how it specifically applies to generalizers based on unification are given in [O'Rorke87b].

The formal definition of a generalized explanation presented in section 3.1 captures O'Rorke's notion of correctness since it requires the global substitution γ to be the most general substitution that makes all equivalent wffs in the explanation structure identical. Since unification produces the most general substitution that makes two wffs identical (as proved in [Robinson65]), the single unification algorithm is guaranteed to produce the correct γ and therefore the correct generalized explanation. In this case, a complete correctness proof for any of the individual algorithms in section 3.2 would require proving that the algorithm is equivalent to the single unification algorithm. As previously mentioned, for the EGGS algorithm, this proof would involve proving the correctness of the unification algorithm in [Charniak80].

3.6. Pruning Explanations for Operationality

Often, the explanation structure for a particular example is too specific to support a reasonably useful generalization. In these cases, the *operationality criterion* [Mitchell86] is met by nodes higher in the explanation tree than the leaves and it is advisable to *prune* units from the explanation structure that are more specific than required for operationality. If this pruning is done prior to generalization as shown in Figure 3.1, it will result in a more abstract generalized explanation which is applicable to a broader range of examples. For example, if the rule for inferring Graspable is removed from the explanation structure shown in Figure 3.3, the following more general (but less operational) definition of Cup is acquired:

$$\text{Bottom}(?y1) \wedge \text{PartOf}(?y1,?x1) \wedge \text{Flat}(?y1) \wedge \text{Graspable}(?x1) \wedge \text{Light}(?x1)$$
$$\wedge\ \text{Concavity}(?y3) \wedge \text{PartOf}(?y3,?x1) \wedge \text{UpwardPointing}(?y3) \rightarrow \text{Cup}(?x1)$$

More appropriate examples of pruning in various domains will be given in the following two chapters and in chapter 11.

Determining the appropriate operationality criterion has been the subject of much discussion in the EBL literature [DeJong86b, Keller87b, Mitchell86, Segre87b, Shavlik87b]. There is clearly a trade-off that must be resolved between operationality and generality. A more general explanation is useful in a larger set of future situations; however, it is also harder to apply in those situations. A more specific explanation, on the other hand, is easier to a apply to future situations; however, it is less applicable. In the long run, it is probably best to retain explanations at several levels of generality as suggested in [Mooney85b] and as done in the PHYSICS101 system [Shavlik87b]. This allows a more specific explanation to be used when it is applicable while still permitting a more general explanation to be used when a more operational one is not available.

A recent suggestion for determining operationality is the one used in ARMS, an EBL system for robotics [Segre87b]. It involves pruning all of the explanation below *shared substructure*. In terms of the representations used here, this approach would prune all nodes below the point where a subgraph of the explanation becomes a tree as opposed a general directed acyclical graph. In other words, it keeps pruning leaves of the explanation until a node is found that supports more than one other node. Although this pruning algorithm may work well for the ARMS domain, it is not a general solution to the problem of determining operationality. Many explanations that support useful generalizations do not have any shared substructure. In fact, most of the examples of explanations presented in the following three chapters are trees and consequently do not have shared substructure. The ARMS approach to pruning would remove the entire explanation in such cases, and consequently miss the opportunity to learn useful new rules and operators.

Therefore, at this point in time, determining which predicates or operators are operational is generally a domain dependent decision. Consequently, the current EGGS system simply has a hook that allows an arbitrary pruning function to be called before an explanation is generalized. Some examples of pruning functions are given in the following two chapters and in chapter 11. In Hirsh's MRS-EBG system [Hirsh87], meta-level logical deduction is used to determine operationality. This approach has the advantage of allowing operationality proofs themselves to be generalized in an explanation-based manner in order to determine the most general operational explanation.

3.7. Integrating Explanation Construction, Pruning, and Generalization

Instead of performing the first three steps in Figure 3.1 sequentially, these steps can often be integrated and performed in an interleaved fashion. As discussed in [DeJong86b, Mooney86c], the EGGS generalization algorithm is easily integrated with the explanation building process by updating the global substitution each time a new rule is added to the evolving explanation. As mentioned earlier, PROLOG-EBG elegantly integrates generalization with the theorem proving process, and MRS-EBG elegantly integrates both of these processes with pruning the explanation for operationality.

Although integrating these processes is aesthetically appealing, there is a price associated with it. For example, the integration of theorem proving and generalization in PROLOG-EBG and MRS-EBG involves unnecessarily generalizing dead-end branches of the search tree which are eventually abandoned and never become part of the final proof. If generalization were postponed until the final proof is available, this useless computation could be avoided. However, as noted in [Hirsh87], integrating generalization and theorem proving is useful when there are multiple possible explanations for the specific example only some of which are operational. In this case, integrated generalization, theorem proving, and operationality checking allows theorem proving to continue until an operational proof is eventually found.

Another problem with integration is that in many cases operationality cannot be determined until the complete explanation is available. When learning by observing the problem solving behavior of an external agent, the eventual goal to be achieved is generally unknown until all of the agent's actions have been observed. However, the pruning algorithm often requires knowledge of the goal. For example, the pruning algorithm used in GENESIS (see chapter 11) needs to know what actions and properties eventually support the achievement of the goal. Consequently, in these situations, pruning must be postponed until the complete explanation has been constructed. If generalization is performed before pruning, the resulting generalization may be too specific since it may incorporate constraints introduced by the pruned parts of the explanation. Therefore, when pruning must be performed after the explanation is complete, generalization and explanation cannot be easily integrated. As discussed in [DeJong86b], if generalization and explanation are integrated, additional constraints introduced by pruned sections of the explanation can later be retracted; however, retracting constraints is very difficult and requires the capabilities of a truth maintenance system (TMS) [Doyle79]. The MA system [O'Rorke87b] is an example of an EBL system that uses a TMS [McAllester82] to retract co-reference constraints; however, this system is very slow compared to simpler systems based on unification. Because of all these problems, explanation construction, pruning, and generalization are performed sequentially in the EGGS system as shown in Figure 3.1.

3.8. Explanation Generalization Versus Chunking and Production Composition

Explanation-based learning of macro-rules and macro-operators is closely related to production system learning mechanisms that compose production rules. The *chunking* process in SOAR [Laird84, Laird86a, Laird86b] and the *knowledge compilation* process of *composition* in ACT* [Anderson83b, Anderson86] are two similar production system learning models. Both processes build macro-productions based on traces of productions produced by the problem solver when solving a particular problem.

Besides the fact that these systems, unlike STRIPS, EBG, and EGGS, do not rely on a logic-based representation, the primary difference lies in the less analytical generalization process. SOAR's generalization algorithm is described in detail in [Rosenbloom86] where it is compared and contrasted to EBG. The generalization process is basically one of changing constants to independent variables. However, due to the difference in representation language, the problem of over-generalization mentioned in section 3.2.1 is avoided. Constants in SOAR come in two types, *identifiers* which are symbols for particular objects, and more meaningful constants such as "5" and "blue." For example, representing the logical assertion Color(β,blue) requires creating an extra identifier for the constant "blue" and using the two assertions Color(β,ζ) and Name(ζ, blue). The generalization process in SOAR changes only identifiers to variables, and since production rules cannot check for particular identifiers, the over-generalization problem is avoided. The cost incurred for avoiding the problem in this manner is an extra distinction in the representation language.

However, the problem of under-generalization mentioned in section 3.2.1 remains. For example, in the SOAR formulation of the Safe-To-Stack problem[6] presented in [Rosenbloom86], the rule learned is:

[6]See section 4.2.1 for a complete specification of this example, which is originally from [Mitchell86].

$$\text{Volume}(x,v) \wedge \text{Density}(x,d) \wedge \text{Name}(y,\text{endtable}) \wedge \text{Product}(v,d,d) \wedge \text{Less}(d,w) \wedge \text{Name}(w,5)$$
$$\rightarrow \text{Safe-To-Stack}(x,y)$$

As noted in [Rosenbloom86], this rule is an under-generalization since it requires the density and the weight of the of object being stacked to be the same (i.e. d). Since the box in the example just happened to have the same weight and density, the simple variablization process requires them to be the same. For the same reason, the initial and final rooms would unnecessarily be required to be same if this technique were applied to the STRIPS example. Retaining such spurious features of the example in the generalization is a basic violation of the explanation-based approach. EBL stipulates that only those constraints required to maintain the validity of the solution should be incorporated in the generalization.

In response to the problem of under-generalization, Rosenbloom and Laird [Rosenbloom86] state that: "If an example were run in which the density and the weight were different, then a rule would be learned to deal with future situations in which they were different.(p. 564)" However, if the more general rule was learned from the original example, this new example could be solved quicker using the learned rule. Also, unless a check is made to remove subsumed rules (i.e. rules that are specializations of existing rules), this solution leaves useless rules lying around which decrease the performance by increasing the number of rules the system must check for application. Rosenbloom and Laird also state that: "The Soar approach to goal regression is simpler, and focuses on the information in working memory rather than the possibly complex patterns specified by the rules....(p. 564)" If the problems with a simple constant to variable generalization algorithm were offset by a marked increased in efficiency of generalization, then perhaps a case could be made for the simpler generalization process. However, as shown in section 3.4, the computational complexity of a unification-based generalization algorithm is linear in the size of the explanation. Since simply tracing through the complete explanation replacing each constant with a variable is also a linear process, the gain in efficiency is at most a constant factor.

Of course, SOAR could probably be modified to include a generalizer that prevents under-generalization. This would require retaining copies of the general parameterized productions (with unique variables) in the production traces produced during problem solving. Generalization would then require constructing the most general set of variable bindings that allows all of the left-hand-sides of general productions to match the right-hand-sides of the general productions that they support in the production trace. A procedure analogous to unification for matching production conditions could be used to produce the required global substitution.

Finally, regarding composition in ACT*, discussions of the underlying generalization process in *production composition* [Anderson82, Anderson83a, Anderson83b, Anderson86] fail to give explicit details of the generalization algorithm. However, the fact that no mention is made of the subtleties of generalization, the limits of simply changing constants to variables, or the use of a generalizer that analyzes variable bindings indicates that a more analytic generalizer is not used. Also, the examples given of composition involving telephone dialing [Anderson82], geometry problem solving [Anderson83a, Anderson83b], and Lisp programming [Anderson83b, Anderson86, Anderson87b] can all be accomplished by simply changing constants to variables. An example of applying EGGS to a geometry example from [Anderson87b] is given in section 4.2.5. However, unlike many of the other examples EGGS has been tested on, the proper generalization for this example could be obtained by simply changing constants to variables.

CHAPTER 4

EGGS: LOGICAL PROOF EXPLANATIONS

In most domains, explanations can be represented as logical proofs that an example meets the definition of a concept or that a combination of operators achieves a goal. Most attempts to formalize explanation-based learning have represented explanations as logical proofs [Hirsh87, Kedar-Cabelli87a, Mitchell86, Prieditis87]. The Cup example presented in the previous chapter is an example of how proofs are represented as explanations in EGGS. This section discusses the EGGS system's ability to construct, understand, generalize, and learn macro-rules from logical proofs. It also presents numerous examples of applying EGGS to various domains.

4.1. Facilities in EGGS Supporting Logical Proof Explanations

The complete EGGS system is equipped with general purpose sub-systems for proving theorems, understanding incomplete proofs, and generating macro-rules. Figure 4.1 illustrates how these components combine with the generalizer to comprise a complete learning system. The theorem prover and proof verifier allow EGGS to construct explanations by either proving assertions itself or supplying reasons and filling in gaps in sketchy proofs provided by the user. These

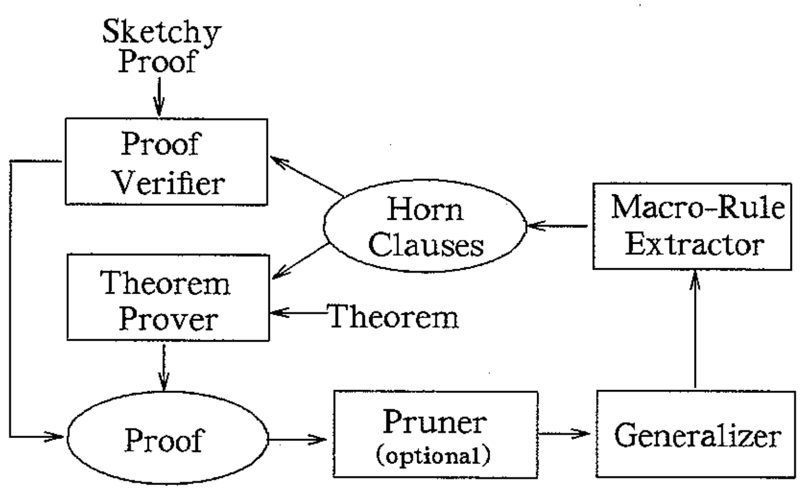


Figure 4.1: EGGS Architecture for Logical Proof Explanations

explanations can then be generalized using the EGGS generalization algorithm, resulting in macro-rules that can be used to construct and understand similar proofs more efficiently.

4.1.1. Theorem Proving

The EGGS' theorem prover, like the theorem prover underlying PROLOG [Clocksin84], is a backward-chaining, depth-first prover for Horn clauses. Specifically, it is a version of the *deductive retriever* given in [Charniak80] modified to return proof trees (explanations) for each answer it retrieves. Instead of automatically returning all possible answers to a query, it uses *generators* [Charniak80, Teitelman83] to produce one answer at a time. Horn clause inference rules for the domain are indexed under the predicate name of their consequent. As in PROLOG, rules for concluding a particular predicate are ordered and are tried in order when searching for a proof. The EGGS' retriever can also be given a depth bound, d, to prevent it from chaining more than d rules deep when attempting to retrieve a particular fact.

As an example of the performance of the EGGS theorem prover, consider the Cup example. Given the facts and rules given in section 3.1 and the goal Cup(Obj1) (or just Cup(?x)) the EGGS retriever produces the proof shown in Figure 3.3.

When a goal is successfully deduced, it is added to the database of facts along with the facts composing its proof. Data dependencies [Charniak80, Doyle79] are maintained between all the facts in the database. These dependencies are used to provide explanations for facts in the database and could also be used to update the a database following the retraction of an assertion. The latter task is one of the primary functions of a *truth maintenance system* [de Kleer86, Doyle79, McAllester82] and is not provided in the current version of EGGS.

4.1.2. Proof Verification

The deductive retriever can also be used to supply reasons and fill in gaps in sketchy proofs given to the system. Sketchy proofs are ordered sequences of subgoals or lemmas to be proven before attempting to prove the ultimate goal. For example, instead of trying to prove Cup(Obj1) directly, the system can be given the following incomplete proof:

Graspable(Obj1)
Liftable(Obj1)
OpenVessel(Obj1)
Q.E.D.: Cup(Obj1)

The system attempts to prove that each step in the proof follows deductively from the initial facts together with the facts deduced in previous steps. When an intermediate step is proven, it is added to the database together with the facts and dependencies in its proof. If it is known that the sketchy proof will not be missing more than d inferences between steps, then the depth bound on the prover can be set to d during verification. The end result of processing a sketchy proof is a complete proof tree for the goal. For the Cup example, the proof constructed by the verifier is the same as the one generated by the theorem prover (see Figure 3.3). More realistic examples of understanding incomplete mathematical proofs are given in sections 4.2.3 and 4.2.5.

Since in the simple Cup example there is only one rule for concluding each predicate, explicitly supplying subgoals does not reduce the search space and is no more efficient than proving the goal directly. Frequently, however, using a sketchy proof can greatly reduce search over undirected theorem proving. For example, let $<l_1, l_2, \ldots l_n>$ be a sketchy proof of l_n from the set of initial facts F. Assume that l_1 can be proven from F with a proof of depth m and that each l_i $(2 \leqslant i \leqslant n)$ can be proven from $F \cup \{l_1 \ldots l_{i-1}\}$ with a proof of depth m (specifically, the path from l_{i-1} to l_i in the proof is of length m). Figure 4.2 shows an abstract specification of the complete proof. Let $d=nm$ be the depth of the complete proof. The verifier can be used to complete this sketchy proof by setting the depth bound to m. The number of nodes searched to prove each lemma will then be: $O(b^m)$, where b is the average branching factor. Since there are $n=d/m$ lemmas, the total search needed to complete the proof is:

$$O\left(\frac{d}{m}b^m\right)$$

As m gets large, b^m becomes the dominant term. This is because by L'Hôpital's rule:

$$\lim_{m \to \infty} \frac{db^m}{m} = \frac{db^m \ln(b)}{1}$$

Consequently, the search time needed to verify the sketchy proof is $O(db^m \ln(b))$. Therefore, for sufficiently large problems, a decrease in the number of inferences between steps in such a sketchy proof can cause an exponential decrease in the time needed to prove the theorem. Since unaided theorem proving is simply a degenerate case of a sketchy proof in which $m=d$, using a sketchy proof can be b^{d-m} times faster than proving the theorem directly.

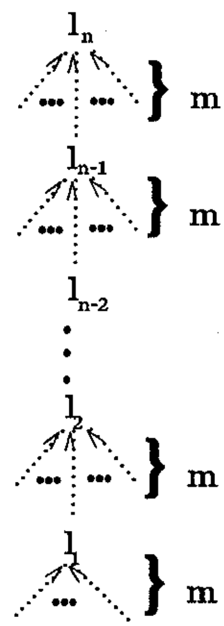


Figure 4.2: Completion of the Sketchy Proof

4.1.3. Learning Macro-Rules

Whether a proof is generated by direct theorem proving or by explaining a sketchy proof, it can be generalized and used to learn a new macro-rule. A macro-rule is obtained from the generalized explanation by taking the goal (root) as the consequent and the leaves as the antecedents. Leaves that are Horn clauses that do not have antecedents but are from the domain theory (e.g. Equal(?x,?x)) represent general facts of the domain, and therefore are not included in the list of antecedents for the macro-rule since they are always true in the given domain. The macro-rule resulting from this process is deductively implied by the existing domain theory since the macro-rule is obtained by chaining together existing domain rules in a particular manner. The macro-rule learned from the Cup example was presented in section 3.1.

After a macro-rule is generated, it is added to the beginning of the current list of rules for inferring instances of the same predicate. Since the backward chaining algorithm tries the rules in order, this insures that the learned rule will be tried first in the future, resulting in performance improvement if the problem can be solved using the learned rule. If the new rule cannot be used to solve the problem, then it eventually resorts to using its previous knowledge to solve the problem. In this case, performance may be degraded since time was wasted attempting to apply the learned rule. Some empirical results as well as further discussion of the effect of rule learning on theorem proving performance is given in chapter 7.

If the EGGS system is used solely as an independent theorem prover which learns macro-rules, it closely resembles a learning PROLOG interpreter, like that described in [Prieditis87]. On the other hand, if it used to analyze sample proofs and learn macro-rules from them, it behaves more like a learning apprentice system [Mitchell85, O'Rorke84]. The important thing to note, however, is that the same generalization algorithm underlies both approaches to learning.

4.2. Logical Proof Examples in Several Domains

Many of the examples of explanation-based learning in the literature can be formalized as learning macro-rules for logical deduction. A deductive formulation of Winston's Cup example and the Suicide example were given in chapter 3. This section presents additional examples from six different domains. It should be noted that since equated wffs are always identical in specific and generalized explanations, equivalent wffs are collapsed in future graphs of explanations.

4.2.1. Safe-To-Stack Example

The Safe-To-Stack example is a simple example introduced in [Mitchell86] to illustrate the EBG explanation-based generalization method. It involves learning an operational rule for when it is safe to stack something on an endtable. The example involves the following facts:

Isa(Obj1,Box), Isa(Obj2,Endtable) Color(Obj1,Red),

Color(Obj2,Blue), Volume(Obj1,1), Density(Obj1,0.1)

The following domain rules can be used to prove that Obj1 can be safely stacked on Obj2:

Lighter(?x,?y) → SafeToStack(?x,?y)

¬Fragile(?y) → SafeToStack(?x,?y)

Volume(?x,?v) ∧ Density(?x,?d) → Weight(?x,?v*?d)

Weight(?x,?w) ∧ Weight(?y,?u) ∧ Less(?w,?u) → Lighter(?x,?y)

Isa(?x,Endtable) → Weight(?x,5)

The specific proof that the EGGS' deductive retriever generated for this problem is shown in Figure 4.3 and the generalized proof is shown in Figure 4.4. The general rule learned from this example is:

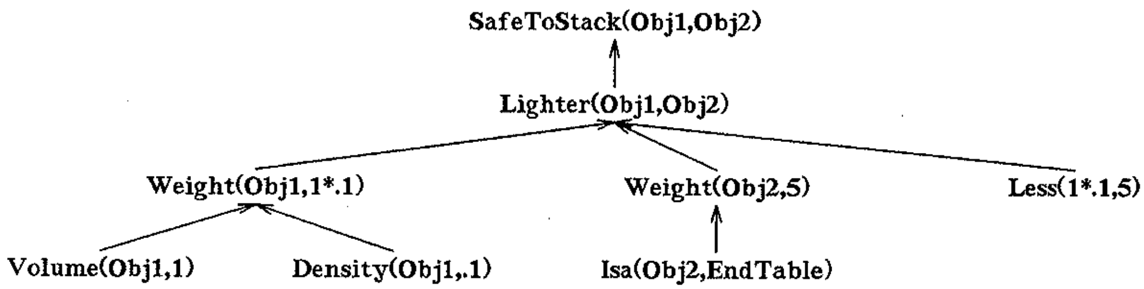


Figure 4.3: Safe-To-Stack Example -- Specific Explanation

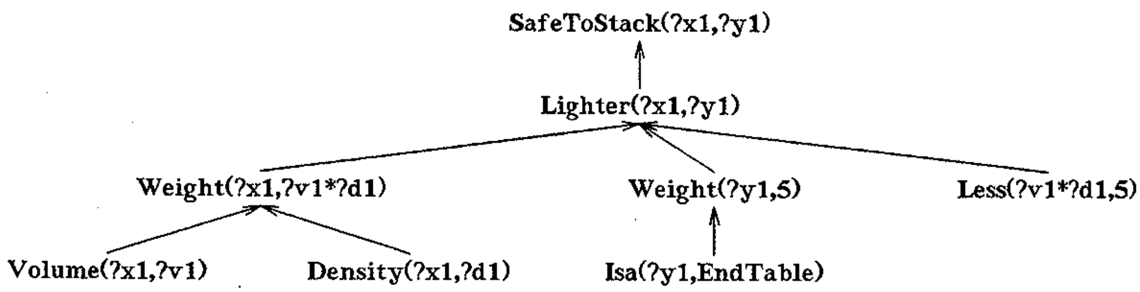


Figure 4.4: Safe-To-Stack Example – Generalized Explanation

Volume(?x1,?v1) $\wedge$ Density(?x1,?d1) $\wedge$ Isa(?y1,Endtable) $\wedge$ Less(v1*d1,5) $\rightarrow$
SafeToStack(?x1,?y1)

This rule can then be used to solve similar problems, such as showing that a book can be safely stacked on an endtable given the facts below:

Isa(Obj1,Book), Isa(Obj2,Endtable), Color(Obj1,Green),
Color(Obj2,Brown), Volume(Obj1,.5), Density(Obj1,2)

4.2.2. LEAP example

The LEAP system [Mitchell85] is a learning apprentice in VLSI design which observes the behavior of a circuit designer. It attempts to learn in an explanation-based fashion by observing and analyzing specific examples of logic design. As an example of learning in this domain, consider the following example taken from [Mitchell85]. Given the task of implementing a circuit that computes the logical function: $(a \vee b) \wedge (c \vee d)$, a circuit designer creates a circuit consisting of three NOR gates like that shown in Figure 4.5. The system attempts to verify that the given circuit actually computes the desired function. The explanation constructed by EGGS' deductive retriever proving that the circuit computes a function that is equivalent to the desired function is shown in Figure 4.6. In this example, the domain knowledge available to the system includes:

Remove Double Negation:	Equiv(?x,?y) → Equiv(¬(¬(?x)),?y)
DeMorgan's Law:	Equiv((¬?x∧¬?y),?a) → Equiv(¬(?x∨?y),?a)
Substitution of Equals:	Equiv(?x,?a) ∧ Equiv(?y,?b) → Equiv(?x∧?y,?a∧?b)
Equivalence Axiom:	Equiv(?x,?x)

The generalized form of this proof is shown in Figure 4.7. The general fact learned from this example is:

Equiv(¬(¬?a∨¬?b), ?a∧?b)

Had generalization been performed by simply changing constants to variables, the result would have been the overly specific rule shown below:

Equiv(¬(¬(?x∨?y)∨¬(?z∨?w)), (?x∨?y)∧(?z∨?w))

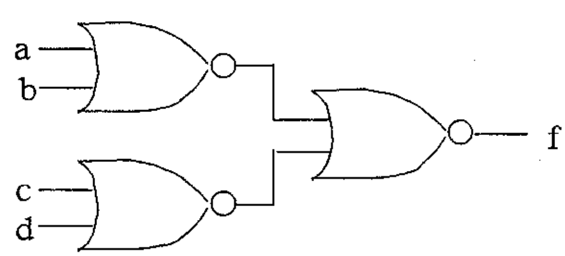


Figure 4.5: Circuit Design Learning Problem

Equiv(¬(¬(a∨b)∨¬(c∨d)),(a∨b)∧(c∨d))

Equiv(¬(¬(a∨b))∧¬(¬(c∨d)),(a∨b)∧(c∨d))

Equiv(¬(¬(a∨b)),a∨b)   Equiv(¬(¬(c∨d)),c∨d)

Equiv(a∨b,a∨b)   Equiv(c∨d,c∨d)

Figure 4.6: LEAP Example – Specific Explanation

Equiv(¬(¬(?a1)∨¬(?b1)),?a1∧?b1)

Equiv(¬(¬(?a1))∧¬(¬(?b1)),?a1∧?b1)

Equiv(¬(¬(?a1)),?a1)   Equiv(¬(¬(?b1)),?b1)

Equiv(?a1,?a1)   Equiv(?b1,?b1)

Figure 4.7: LEAP Example -- Generalized Explanation

As a result of the explanation-based approach, the resulting generalization is not sensitive to the fact that the first stage of the circuit involved two NOR gates.

For example, the generalization would support using two NAND gates and a NOR gate to AND four inputs together as shown in Figure 4.8. The rule learned from the previous example is capable of aiding in both the design and verification of this circuit. Given the query: Equiv(?x, (a∧b) ∧ (c∧d))) the deductive retriever "designs" the circuit shown in Figure 4.8, and given the query Equiv(¬(¬(a∧b) ∨ ¬(c∧d)), (a∧b) ∧ (c∧d))) it verifies it. The learned rule allows both of these tasks to be done in one step. This example nicely illustrates how a learned rule can increase the efficiency of both the construction and understanding of solutions to similar problem.

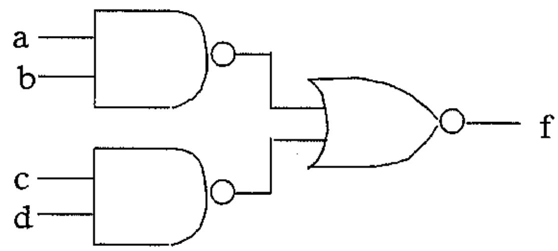


Figure 4.8: Circuit Design Test Problem

4.2.3. MA Example

Another explanation-based learning system in the domain of logic is MA [O'Rorke84, O'Rorke87b] which learns proof schemata from sample natural deduction proofs. When the system cannot complete a proof for a particular theorem, a teacher steps in and completes the proof. MA then generalizes the teacher's proof in an explanation-based manner to generate a proof schema that can be used to solve future problems. Consider a variant of the example discussed in [O'Rorke84] of proving a particular case of the law of excluded middle: NIL⇒(P∧Q)∨¬(P∧Q) (i.e. (P∧Q)∨¬(P∧Q) can be deduced from the empty set of assumptions). The natural deduction proof the deductive retriever generates for this example is shown in Figure 4.9. The following rules of natural deduction [Manna74] are employed in this proof:

Assumption Axiom:	(?x . ?y)⇒?x
Or Introduction:	?x⇒?y → ?x⇒?y∨?z
Or Introduction:	?x⇒?y → ?x⇒?z∨?y
Elimination Of Assumption:	(?x . ?y)⇒?z ∧ (¬?x . ?y)⇒?z → ?y⇒?z

The expression ?x⇒?y means that the wff ?y is deducible from the list of assumptions (wffs) ?x. LISP *dot notation* is used to represent lists of assumptions. The generalized proof EGGS generates for this example is shown in Figure 4.10. From a specific instance of proving (P∧Q)∨¬(P∧Q) from no assumptions, a general proof is learned for proving the disjunction of any wff and its negation from any set of assumptions. Notice again that simply changing the constants P and Q to variables would have resulted in an under-generalization. The more general fact learned by EGGS allows it to solve the test problem: NIL⇒P∨¬P in one step.

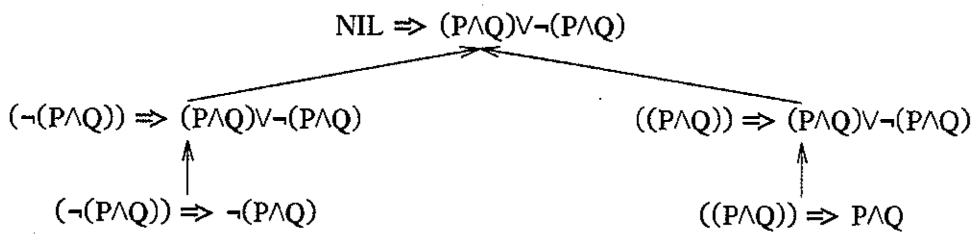


Figure 4.9: MA Example – Specific Explanation

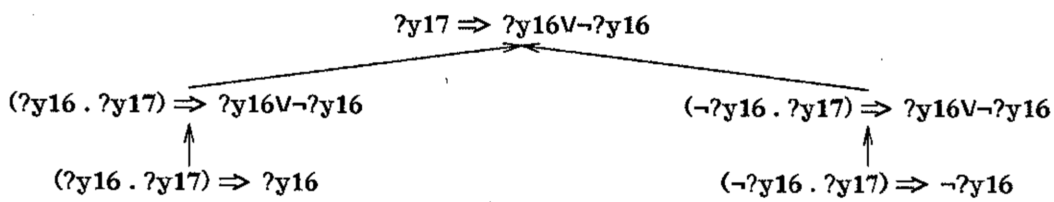


Figure 4.10: MA Example – Generalized Explanation

The same proof and generalization shown in Figures 4.9 and 4.10 result from understanding the following sketchy proof:

((P∧Q)) ⇒ (P∧Q) ∨ ¬(P∧Q)

(¬(P∧Q)) ⇒ (P∧Q) ∨ ¬(P∧Q)

Q.E.D.: NIL ⇒ (P∧Q) ∨ ¬(P∧Q)

If the depth bound on the deductive retriever is set to two, explaining this proof takes only 4.5 CPU seconds, compared to proving the theorem directly which takes 9 CPU seconds with a depth bound of three, 18.5 CPU seconds with a depth bound of five, and 759 CPU seconds with a depth bound of ten! This simply demonstrates that, as expected, understanding a proof can be much more efficient than independently discovering one.

4.2.4. LEX2 Example

LEX2 [Mitchell83] is an explanation-based learning system in the domain of integration problem solving. Unlike the other systems addressed in this paper which learn new inference rules or plans, LEX learns search control knowledge. Specifically, it learns heuristics for when to apply particular integration operators by determining, in an explanation-based manner, why the application of a particular operator led to the solution of a specific problem. The formulations of LEX2's approach given in [Mitchell86] and [Hirsh87] use a set of logical inference rules to prove that a particular operator was "useful" in solving a particular problem. However, macro-rules for solving a particular class of integration problems can also be learned in an explanation-based manner, and the conditions that allowed each operator to lead to the solution are easily extracted from the generalized explanation. For example, consider solving the following problem discussed in [Mitchell86]: $\int 3x^2dx$. The solution to this problem can be formulated as logical inference using the following inference rules:

$$\text{Constant}(?r) \wedge ?r \neq -1 \rightarrow \text{Solution}(\int ?x^{?r}d?x,\ ?x^{?r+1}/(?r+1))$$

$$\text{Constant}(?c) \wedge \text{Solution}(\int ?fd?x,\ ?z) \rightarrow \text{Solution}(\int ?c*?fd?x,\ ?c*?z)$$

$$\text{Numberp}(?x) \rightarrow \text{Constant}(?x)$$

The expression Solution(?x,?y) means ?y is the solution to the problem ?x. The proof EGGS generated as the solution to the given problem is shown in Figure 4.11, and the generalized proof is shown in Figure 4.12. The general rule learned from this example is:

$$\text{Numberp}(?c1) \wedge \text{Numberp}(?r1) \wedge ?r1 \neq -1 \rightarrow \text{Solution}(\int ?c1*?x1^{?r1}d?x1,\ ?c1*(?x1^{?r1+1}/(?r1+1)))$$

This rule can then be used to immediately solve similar problems like: $\int 5x^3dx$. The conditions that allowed the first operator to lead to a solution are also implicitly represented in this rule. The problem must match the form in the conclusion: $\int ?c1*?x1^{?r1}d?x1$ and satisfy the antecedents of the rule: Numberp(?c1), Numberp(?r1), and ?r1≠-1. In a problem with more steps, the "useful" conditions for other operators in the solution can be obtaining by *pruning* (i.e. removing from the explanation structure) the steps for all the previous operators in the solution prior to generalizing the explanation structure. This will create a macro-rule for the last n steps, and the "useful" conditions for the first operator in this macro-rule can then be extracted as described above.

Unlike the process for learning integration problem solving described in [Mitchell86], this method obtains both a general macro-rule and heuristic conditions from a single explanation using a single generalizing mechanism. The method described in [Mitchell86] learns only search

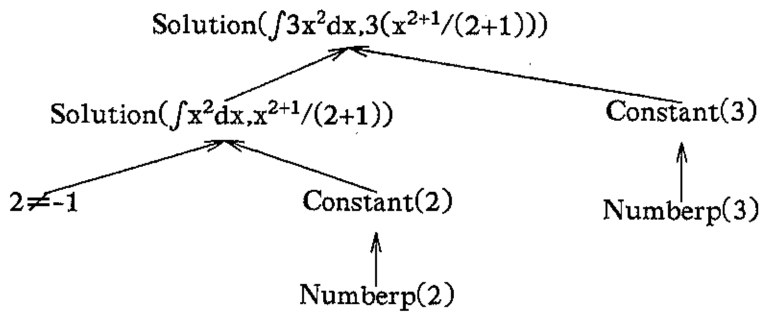


Figure 4.11: LEX2 Example – Specific Explanation

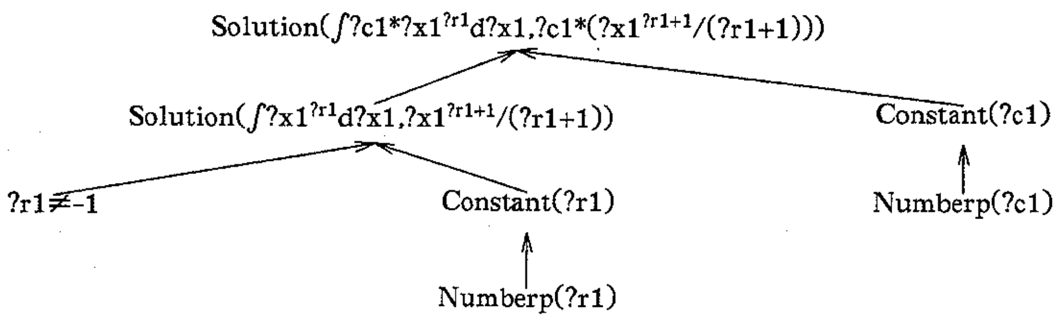


Figure 4.12: LEX2 Example – Generalized Explanation

heuristics and uses the EBG method as well as a separate regression step using rules of the domain theory to regress operators through problem states. If the problem solving operators are explicitly represented in the rules of the domain theory, this separate regression step is unnecessary. Like the approach presented here, the formulation given in [Hirsh87], explicitly represents operators in the domain rules in order to eliminate the unnecessary regression step. However, like Mitchell *et. al.'s* formulation, Hirsh's approach learns only search heuristics for individual operators and not macro-rules. The approach presented here learns both heuristics and macro-rules; however, it requires a specialized technique for extracting the heuristics from the generalized explanation.

4.2.5. Geometry Example

Another mathematical domain to which EGGS has been applied is proving theorems in elementary Euclidian geometry. J. R. Anderson has used his production system learning models ACT* [Anderson83a, Anderson83b] and PUPS [Anderson87b]. to model human learning in this domain. EGGS is capable of learning general geometry proof rules analogous to the productions produced by Anderson's *composition* process. These learned rules can then be used to solve similar problems more efficiently in the future.

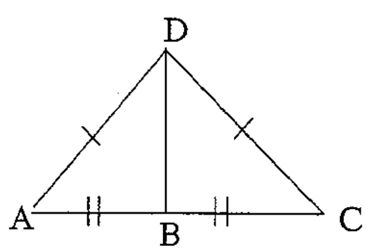
For example, consider the problem shown in Figure 4.13 which was taken from [Anderson87b]. The domain rules needed to solve this problem are:

Reflexivity:	$?x \cong ?x$
Side Side Side:	$\overline{?a?b} \cong \overline{?x?y} \wedge \overline{?b?c} \cong \overline{?y?z} \wedge \overline{?c?a} \cong \overline{?z?x} \rightarrow \Delta ?a?b?c \cong \Delta ?x?y?z$
Line Axiom:	$\overline{?x?y} = \overline{?y?x}$
Equality Axiom:	$?x = ?y \wedge ?x \cong ?z \rightarrow ?y \cong ?z$
Equality Axiom:	$?x = ?y \wedge ?z \cong ?x \rightarrow ?z \cong ?y$

Given these rules, EGGS' backward chaining inference engine constructs the proof shown in Figure 4.14. The generalized proof produced by EGGS is shown in Figure 4.15. The general composed rule learned from this problem solution is:

$$\overline{?a6?z3} \cong \overline{?z3?b5} \wedge \overline{?a6?a5} \cong \overline{?a5?b5} \rightarrow \Delta ?a6?a5?z3 \cong \Delta ?b5?a5?z3$$

This rule is analogous to the composed production learned by ACT* since the simple method of replacing each constant with an independent variable produces the correct result for this particular



GIVEN: $\overline{AB} \cong \overline{BC}$
$\overline{AD} \cong \overline{DC}$
$\overline{ABC}$

PROVE: $\Delta ABD \cong \Delta CBD$

Figure 4.13: Geometry Learning Problem

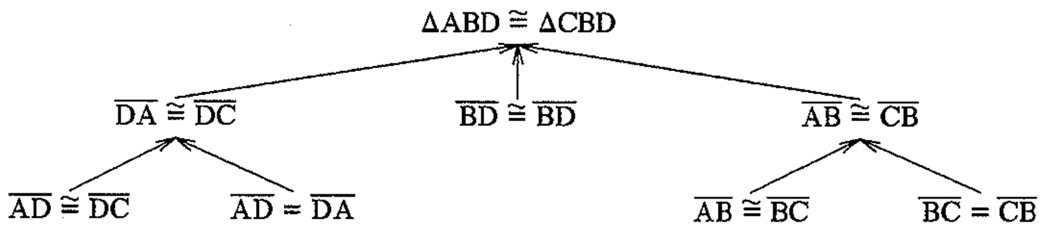


Figure 4.14: Geometry Example -- Specific Explanation

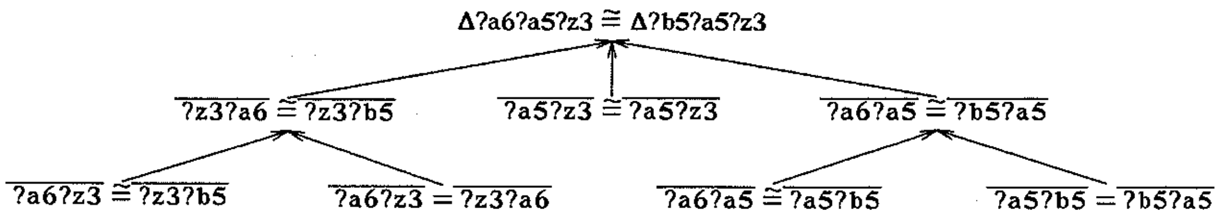
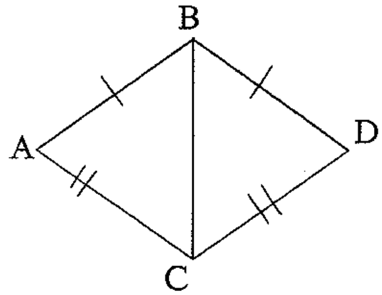


Figure 4.15: Geometry Example -- Generalized Explanation

example. However, as many of the other examples illustrate, this simple generalization method frequently produces overly-specific generalizations.

The rule learned by EGGS is capable of directly solving similar problems such as the one shown in Figure 4.16. Since the solution to the previous problem did not depend on the bottom three points being colinear, the general proof used for the previous problem can also be used for this problem.

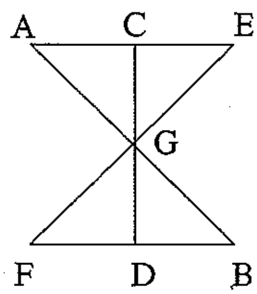
An additional geometry problem, shown in Figure 4.17 (taken from [Anderson83a] and originally from [Jurgenson75]), is difficult enough so that the inherent limits of the machine (a Xerox 1108) prevents the EGGS theorem prover from solving the problem. After 28 minutes of run time with a depth bound of nine (the depth of the shortest known proof), the system's storage is exhausted and a proof has still not been found. However, understanding the following sketchy proof with a depth bound of three takes only 78 seconds of CPU time.



GIVEN: $\overline{AB} \cong \overline{BD}$
$\overline{AC} \cong \overline{CD}$

PROVE: $\Delta ABC \cong \Delta DBC$

Figure 4.16: Geometry Test Problem



GIVEN: $\overline{AGB}$ $\overline{FGE}$
$\overline{ECA}$ $\overline{BDF}$ $\overline{DGC}$
G is midpoint of $\overline{AB}$
G is midpoint of $\overline{CD}$

PROVE: G is midpoint of $\overline{EF}$

Figure 4.17: Difficult Geometry Example

$\Delta CGA \cong \Delta DGB$

$\angle GAE \cong \angle GBF$

$\Delta EGA \cong \Delta FGB$

Q.E.D.: Midpoint(G,$\overline{EF}$)

This example clearly demonstrates the importance of being able to learn from hints and the observed problem solving behavior of other agents. Explanation-based learning from observation allows a system to learn how to solve problems that it was previously unable to solve due to inherent computational resource limitations.

4.2.6. Deductive STRIPS Planning Example

As described in [Nilsson80] and [Kowalski79], traditional blocks world planning can be formulated as logical deduction. The following discussion will follow what Nilsson refers to as "Kowalski's formulation" of planning as deduction. In this formulation, states and actions are treated as terms instead of predicates. Holds(?f,?s) means that the fact ?f holds in the state ?s. The term do(?a,?s) represents the state achieved by performing the action ?a in state ?s. Poss(?s) means that ?s is a possible state, i.e. one that can be reached from the current state. Pact(?a,?s) means that it is possible to perform action ?a in state ?s, i.e. all of ?a's preconditions hold in state ?s. Diff(?x,?y) means that ?x and ?y are different, distinct terms. Finally, Achieve(?f,?s) means that fact ?f is achieved in state ?s, i.e. ?f holds in ?s and ?s is a reachable state. A set of axioms for handling examples from the STRIPS domain [Fikes72] is given below.

Holds(?f,?s) ∧ Poss(?s) → Achieve(?f,?s)

Pact(?a,?s) ∧ Poss(?s) → Poss(do(?a,?s))

Effect Axioms:

Holds(inroom(?a,?l),do(gothru(?a,?d,?k,?l),?s))

Holds(inroom(?b,?l),do(pushthru(?a,?b,?d,?k,?l),?s))

Holds(inroom(?a,?l),do(pushthru(?a,?b,?d,?k,?l),?s))

Frame Axioms:

Holds(?v,?s) ∧ Diff(?v,inroom(?a,?k)) ∧ Diff(?v,inroom(?b,?k)) →
  Holds(?v,do(pushthru(?a,?b,?d,?k,?l),?s))

Holds(?v,?s) ∧ Diff(?v,inroom(?a,?k)) → Holds(?v,do(gothru(?a,?d,?k,?l),?s))

Precondition Axioms:

Agent(?a) ∧ Holds(inroom(?a,?k),?s) ∧ Holds(connects(?d,?k,?l),?s) →
  Pact(gothru(?a,?d,?k,?l),?s)

Agent(?a) ∧ Holds(inroom(?a,?k),?s) ∧ Holds(inroom(?b,?k),?s) ∧
  Holds(connects(?d,?k,?l),?s) → Pact(pushthru(?a,?b,?d,?k,?l),?s)

Miscellaneous Axioms:

ATOM(?x) ∧ NOT(EQ(?x,?a)) → Diff(?x,?a)

Diff(?x,?a) → Diff((?x . ?y),(?a . ?b))

Diff(?y,?b) → Diff((?x . ?y),(?a . ?b))

Holds(connects(?d,?l,?k),?s) → Holds(connects(?d,?k,?l),?s)

Given the following facts describing the initial state of the world shown in Figure 3.6.

Poss(S0), Agent(Robot), Holds(inroom(Robot,Room1),S0),

Holds(inroom(Box,Room2),S0), Holds(connects(Door1,Room1,Room2),S0)

and the goal: Achieve(inroom(Box,Room1),?s), the deductive retriever constructs the proof shown in Figure 4.18. The generalized proof for this example is shown in Figure 4.19. The explanation has been pruned to remove the rule: Not(Eq(?b1,?a14)) ∧ Atom(?b1,?a14) → Diff(?b1,?a14). In order to increase generality, the pruning algorithm removes instances of this general rule that add additional antecedents to the macro-rule. The antecedents of the rule: Not(Eq(Connects,inroom)) ∧ Atom(Connects) → Diff(connects,inroom) are general facts of the domain and therefore are not added to the antecedents of the macro-rule. Consequently, this rule does not detract from the generality of the final result and is not pruned. The macro-rule learned from the final generalized explanation is:

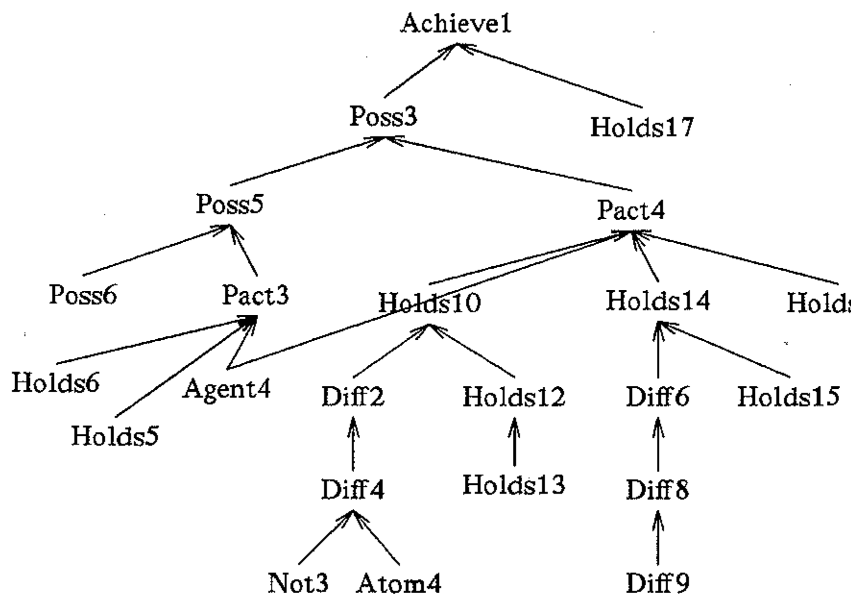
Poss(?s8) ∧ Holds(connects(?d5,?k5,?k3),?s8) ∧ Holds(inroom(?a14,?k5),?s8) ∧ Agent(?a14) ∧
Holds(connects(?d3,?l3,?k3),?s8) ∧ Diff(?b1,?a14) ∧ Holds(inroom(?b1,?k3),?s8) →
Achieve(inroom(?b1,?l3),do(pushthru(?a14,?b1,?d3,?k3,?l3),do(gothru(?a14,?d5,?k5,?k3),?s8)))

Notice that this generalization allows the destination of the box (?l3) to be different from the room in which the robot started (?k5). As discussed earlier, a generalization produced by simply changing each constant to an independent variable would require them to be the same like they were in the example. On the other hand, the macro-rule learned by EGGS can be used to solve the following problem:

Goal: Achieve(inroom(Box,Room3),?s)

Given: Poss(S0), Agent(Robot), Holds(inroom(Box,Room2),S0)
Holds(inroom(Robot,Room1),S0), Holds(connects(Door1,Room1,Room2),S0)
Holds(connects(Door2,Room3,Room2),S0)

In addition to being able to generate plans, this formulation can also easily be used to verify that a particular plan works. For example, the STRIPS plan can be verified instead of generated by simply proving the following assertion:



Achieve1	Achieve(inroom(?b1,?l3),do(pushthru(?a14,?b1,?d3,?k3,?l3), do(gothru(?a14,?d5,?k5,?k3),?s8)))
Poss3	Poss(do(pushthru(?a14,?b1,?d3,?k3,?l3),do(gothru(?a14,?d5,?k5,?k3),?s8)))
Poss5	Poss(do(gothru(?a14,?d5,?k5,?k3),?s8))
Poss6	Poss(?s8)
Pact3	Pact(gothru(?a14,?d5,?k5,?k3),?s8)
Holds6	Holds(connects(?d5,?k5,?k3),?s8)
Holds5	Holds(inroom(?a14,?k5),?s8)
Agent4	Agent(?a14)
Pact4	Pact(pushthru(?a14,?b1,?d3,?k3,?l3),do(gothru(?a14,?d5,?k5,?k3),?s8))
Holds10	Holds(connects(?d3,?k3,?l3),do(gothru(?a14,?d5,?k5,?k3),?s8))
Diff2	Diff(connects(?d3,?k3,?l3),inroom(?a14,?k5))
Diff4	Diff(connects,inroom)
Not3	Not(Eq(connects,inroom))
Atom4	Atom(connects)
Holds12	Holds(connects(?d3,?k3,?l3),?s8)
Holds13	Holds(connects(?d3,?l3,?k3),?s8)
Holds14	Holds(inroom(?b1,?k3),do(gothru(?a14,?d5,?k5,?k3),?s8))
Diff6	Diff(inroom(?b1,?k3),inroom(?a14,?k5))
Diff8	Diff(?b1(?k3),?a14(?k5))
Diff9	Diff(?b1,?a14)
Holds15	Holds(inroom(?b1,?k3),?s8)
Holds16	Holds(inroom(?a14,?k3),do(gothru(?a14,?d5,?k5,?k3),?s8))
Holds17	Holds(inroom(?b1,?l3),do(pushthru(?a14,?b1,?d3,?k3,?l3), do(gothru(?a14,?d5,?k5,?k3),?s8)))

Figure 4.19: Logic STRIPS Example -- Generalized Explanation

Achieve(inroom(Box,Room1),do(pushthru(Robot,Box,Door1,Room2,Room1),
do(gothru(Robot,Door1,Room1,Room2),S0)))

Verifying plans is useful for constructing explanations for observed plans and thereby learning general plans from observed behavior.

CHAPTER 5

EGGS: REWRITE EXPLANATIONS

Many mathematical problems can be more concisely represented and solved using *rewrite rules* [Bundy83] instead of logical deduction. Domains that involve the manipulation of mathematical formulae generally require rewriting subterms of an expression. Rewriting subterms using logical deduction requires the representation and skillful application of equality axioms. On the other hand, the type of equality reasoning generally required in most mathematical domains is a natural part of a mechanism that applies rewrite rules. Consequently, a rewrite system can be a more efficient way of solving particular types of problems. The empirical data presented in chapter 7 shows that the EGGS rewrite system can be about one and a half to four times faster than the EGGS theorem prover at solving the same problem.

Like the previous section on logical proofs, this section discusses the EGGS system's ability to construct, generalize, and learn macro-rules from specific rewritings. It also presents a few examples of applying EGGS to various domains that use rewritings as explanations.

5.1. Rewritings as Explanations: The LEAP Example

Like logical proofs, chains of rewrite rule applications can be represented as explanations. Each rewritten expression is a pattern in the explanation and the chaining together of two rules is represented by a unification between the right-hand-side (RHS) of the first rule and the left-hand-side (LHS) of the second. For example, consider solving the LEAP example discussed in section 4.2.2 by rewriting the expression $\neg(\neg(a \vee b) \vee \neg(c \vee d))$ to $(a \vee b) \wedge (c \vee d)$ using the following rewrite rules:

$$\neg\neg ?x \rightarrow ?x$$

$$\neg(?x \vee ?y) \rightarrow \neg ?x \wedge \neg ?y$$

The explanation, explanation structure, and generalized explanation for this example are shown in Figure 5.1, Figure 5.2, and Figure 5.3 respectively. Notice that if a rule rewrites only a subterm of an expression, "dummy variables" are added to the pattern in the explanation structure to fill out the expression so that it can be unified with the previous expression. LISP list notation is used in the illustration of the explanation structure since the technique used to add dummy variables relies on the underlying list representation. For example, after applying DeMorgan's Law in the LEAP problem, the rule for eliminating a double negation is used to rewrite the first term in the resulting

$$
\begin{array}{c}
\neg(\neg(a \vee b) \vee \neg(c \vee d)) \\
\downarrow \\
\neg(\neg(a \vee b)) \wedge \neg(\neg(c \vee d)) \\
\downarrow \\
(a \vee b) \wedge \neg(\neg(c \vee d)) \\
\downarrow \\
(a \vee b) \wedge (c \vee d)
\end{array}
$$

Figure 5.1: LEAP Rewrite -- Specific Explanation

```
(not (or ?x1 ?y1))
        |
        v
(and (not ?x1) (not ?y1))
       |||
(?f1 (not (not ?x2)) . ?f2)
        |
        v
(?f1 ?x2 . ?f2)
       |||
(?f3 ?f4 (not (not ?x3)) . ?f5)
        |
        v
(?f3 ?f4 ?x3 . ?f5)
```

Figure 5.2: LEAP Rewrite -- Explanation Structure

$$\neg(\neg(?f4) \vee \neg(?x3))$$
$$\downarrow$$
$$\neg(\neg(?f4)) \wedge \neg(\neg(?x3))$$
$$\downarrow$$
$$?f4 \wedge \neg(\neg(?x3))$$
$$\downarrow$$
$$?f4 \wedge ?x3$$

Figure 5.3: LEAP Rewrite -- Generalized Explanation

conjunction. In order to allow the LHS of this rule to unify with the RHS of the instance of DeMorgan's Law in the explanation structure, this rule is padded with the dummy variables ?f1 and ?f2. During generalization, this unification results in the following substitution:

{and/?f1, not(?x2)/?x1, (not(?y1))/?f2}

Next, the rule for eliminating a double negation is used to rewrite the second term in the conjunction and this instance of the rule is padded with the dummy variables ?f3, ?f4, and ?f5 in order to allow its LHS to unify with the previous instance. During generalization, this unification results in the following substitution:

{and/?f3, ?f4/?x2, not(?x3)/?y1, NIL/?f5}

Applying the composition of these two substitutions to the explanation structure results in the generalized explanation shown in Figure 5.3.

5.2. Facilities in EGGS Supporting Rewrite Explanations

Figure 5.4 illustrates the components in EGGS that use and learn rewrite rules. The rewrite-rule engine allows EGGS to construct rewrite-rule explanations by solving problems itself. These explanations can then be generalized, resulting in macro-rules that can be used to solve similar problems more efficiently. A system for understanding sketchy rewritings provided by the user would also allow the system to operate as a learning apprentice. Although the current EGGS system is not equipped with such a sub-system, it would not be difficult to construct one. An example of such an understanding system is discussed in [Bennett86].

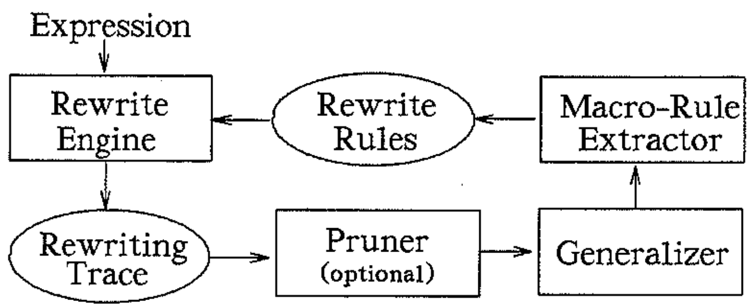


Figure 5.4: EGGS Architecture for Rewrite Rule Explanations

5.2.1. Rewriting Expressions

There are a number of techniques for using rewrite rules to solve problems. If a set of rewrite rules is *canonical* [Bundy83], any sequence of legal rewrite rule applications will eventually convert an expression to canonical form. If a rule set is not canonical, it is necessary to search through the space of possible rewritings until an expression is found that meets a specified criterion.

EGGS is equipped with both a deterministic rewrite system for applying canonical rule sets and a backtracking rewrite system for rule sets that are not canonical. The deterministic system simply applies a rule to rewrite the current expression or one of its sub-expressions. This process is continued until no more rules apply. The backtracking system searches possible rewritings in either a depth-first or breadth-first manner until a supplied goal predicate is satisfied.

In both cases, there are various ways of choosing which rule to apply at each step. For the deterministic system, this choice determines the length of the eventual path to the canonical form. For the backtracking system, this choice also effects the amount of search required. In his book, Bundy presents two methods for choosing which expression to rewrite. The first is to rewrite the leftmost/innermost subexpression that can be rewritten. This method is called *innermost* or *call by value*. The second is to rewrite the leftmost/outermost subexpression that can be rewritten. This method is called *outermost* or *call by name*. In order to allow learned macro-rules to have priority over solving problems from scratch, a third technique called *rule ordered* is used in EGGS. In this technique, the rewrite rules are ordered and the sub-expression that matches the LHS of the the highest numbered rule is rewritten.

The three examples presented in this section all use the deterministic rewrite system. Complete rule sets capable of solving any problem in the domains discussed would not be canonical and would require using the backtracking system. For the LEAP example discussed above, the deterministic rewrite engine produces the rewriting shown in Figure 5.1.

5.2.2. Learning Rewrite Macro-Rules

After the solution to a specific rewriting problem is determined by the rewrite rule engine, the EGGS generalization algorithm is used to produce a generalized explanation like that shown in Figure 5.3. A rewrite macro-rule is easily extracted from the generalized explanation by taking the first expression in the chain as the LHS and the last expression as the RHS. The new rewrite macro-rule that is learned from the LEAP example is:

$\neg(\neg?f4 \vee \neg?x3) \rightarrow ?f4 \wedge ?x3$

A learned macro-rule is added to the beginning of the list of rewrite rules so that, due to the rule ordered method, it is tried first when solving subsequent problems. As a result, performance is improved on subsequent problems that can be solved using the macro-rule. However, as with logical inference macro-rules, unsuccessfully trying to apply the new rule will slightly degrade performance on other problems. Some empirical data on the effect of learning on rewriting performance is given in chapter 7.

5.3. Rewrite Examples

This section presents two more examples of rewrite-rule explanations in addition to the LEAP example discussed above. They include a rewrite formulation of the LEX example presented earlier and an example from the domain of solving algebraic equations.

5.3.1. Rewrite Version of the LEX2 Example

Examples from LEX2's domain of solving integrals can also be more concisely formulated as rewritings. Consider the example discussed earlier of solving the integral: $\int 7x^2$. The following two rewrite rules are needed to solve this problem:

$\int(?c*?f)d?x \rightarrow ?c*\int ?f?d?x$ if $\neg$OccursIn(?x,?c)

$\int ?x^{?r}d?x \rightarrow ?x^{?r+1}/(?r+1)$ if $?r \neq -1 \wedge \neg$OccursIn(?x,?r)

The *if* conditions attached to the rewrite rules are additional conditions that must be met for the rewrite rule to apply. The specific and generalized explanations generated for this problem by the

EGGS rewriting system and generalizer are shown in Figure 5.5 and Figure 5.6 respectively. The rewrite macro-rule learned from this example is:

$$\int(?f3*(?x2^{?r1}))d?x2 \rightarrow ?f3*((?x2^{?r1+1})/(?r1+1))$$
$$\text{if } \neg \mathrm{OccursIn}(?x2,?f3) \wedge ?r1 \neq -1 \wedge \neg \mathrm{OccursIn}(?x2,?r1)$$

This rule can then be used to solve additional problems such as $\int 5x^3$ more efficiently.

5.3.2. Equation Solving Example

EGGS has also been tested in the domain of solving algebraic equations. The LP system [Silver83, Silver86] learns by analyzing individual solutions to equations; however, it learns very abstract "strategies" which are not guaranteed to solve problems to which they can be applied.

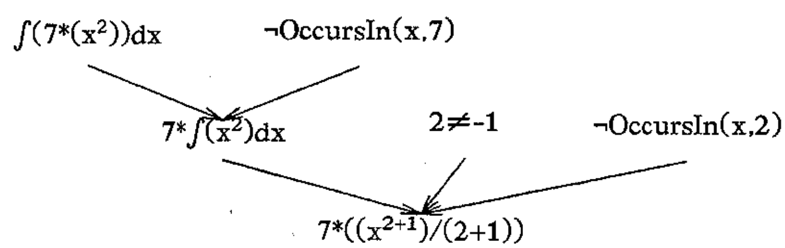


Figure 5.5: LEX2 Rewrite Example -- Specific Explanation

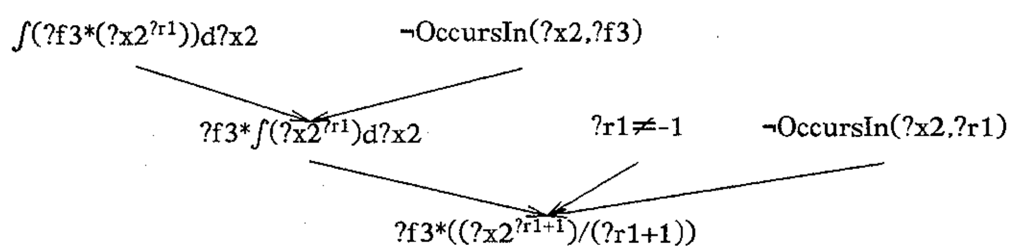


Figure 5.6: LEX2 Rewrite Example -- Generalized Explanation

These strategies are then executed in a flexible manner when attempting to solve future problems. EGGS, on the other hand, learns more specific rewrite rules which are guaranteed to solve a particular class of problems.

The background knowledge used in this domain is also most conveniently given in terms of rewrite rules. The following set of rules can be used to solve the equation: $\log_e(x+1) + \log_e(x-1) = c$ (from [Bundy83]).

$\log_{?b}?u=?v \rightarrow ?u=?b^{?v}$ if OccursIn(x,?u) $\wedge$ NotOccurIn(x,?v)

$?u-?v=?z \rightarrow ?u=?z+?v$ if OccursIn(x,?u) $\wedge$ NotOccurIn(x,?v)

$?u^2=?v \rightarrow ?u=\mathrm{sqrt}(?v)$ if OccursIn(x,?u) $\wedge$ NotOccurIn(x,?v))

$(?u+?v)*(?u-?v) \rightarrow (?u^2)-(?v^2)$ if OccursIn(x,?u)

$\log_{?w}?u+\log_{?w}?v \rightarrow \log_{?w}(?u*?v)$ if OccursIn(x,?u) $\wedge$ OccursIn(x,?v) $\wedge$ NotOccurIn(x,?w)

The *if* conditions attached to the rules constitute control information which assure that the application of a rule is a step towards the solution. This control information forms a part of the explanation and is learned along with the body of the new compiled rewrite rule. These conditions are proven deductively using the EGGS theorem prover and the following inference rules:

1) OccursIn(?x,?x)

2) OccursIn(?x,?y) $\rightarrow$ OccursIn(?x,(?y . ?z))

3) OccursIn(?x,?z) $\rightarrow$ OccursIn(?x,(?y . ?z))

4) Atom(?y) $\wedge$ $\neg$Eq(?x,?y) $\rightarrow$ NotOccurIn(?x,?y)

5) NotOccurIn(?x,?y) $\wedge$ NotOccurIn(?x,?z) $\rightarrow$ NotOccurIn(?x,(?y . ?z))

The specific explanation for the solution to sample equation is given in Figure 5.7. If this explanation is generalized without pruning, the following rule is extracted from the generalized explanation:

$\log_{?b1}(x+?v3)+\log_{?b1}(x-?v3)=?v2 \rightarrow x=\mathrm{sqrt}((?b1^{?v2})+(?v3^2))$

if Atom(?b1) $\wedge$ $\neg$Eq(x,?b1) $\wedge$ Atom(?v2) $\wedge$ $\neg$Eq(x,?v2) $\wedge$ Atom(?v3) $\wedge$ $\neg$Eq(x,?v3)

Maintaining the complete structure of the original explanation has required ?b1, ?v2, and ?v3 to be atoms as they were in the example. That is, in the example, it was proved that ?b1 (the base of the logarithm) did not contain x because it was an Atom that was not Eq to x. To increase the generality of learned macro-rules, a special pruning procedure for this domain automatically prunes instances of inference rules 1 and 4 from the explanation structure. The generalized explanation produced from the pruned explanation is shown in Figure 5.8. The new rewrite-rule learned from this generalized explanation is:

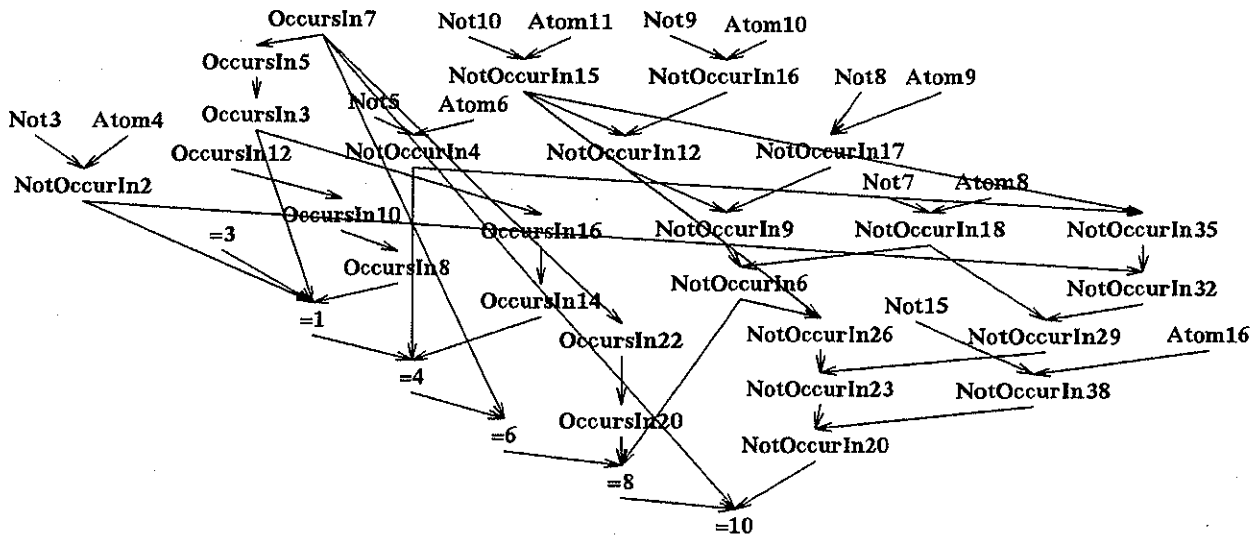
$\log_{?b1}(?y8+?v3)+\log_{?b1}(?y8-?v3)=?y2 \rightarrow ?y8=\mathrm{sqrt}((?b1^{?y2})+(?v3^2))$

if NotOccurIn(x,?b1) $\wedge$ NotOccurIn(x,?y2) $\wedge$ NotOccurIn(x,?v3) $\wedge$ OccursIn(x,?y8)

This rule is at the appropriate level of generality to cover a relatively large class of problems. For example, this rule can now be used to solve the following test example in one step.

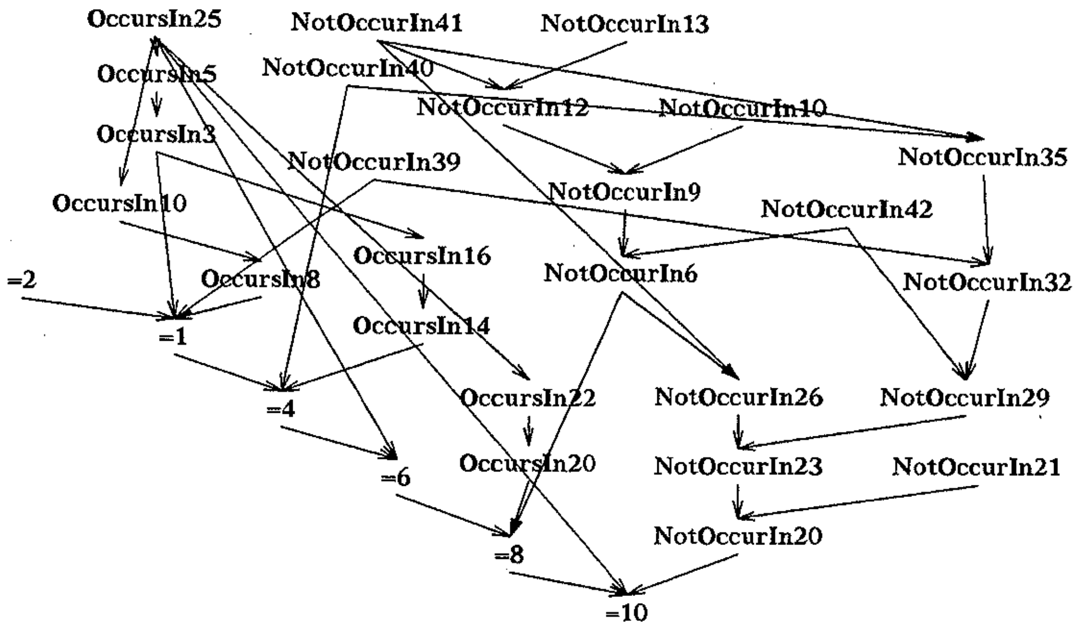
$\log_{3*e}(x+2*a) + \log_{3*e}(x-2*a)=c+d$

$x=\mathrm{sqrt}((3*e)^{c+d}+(2*a)^2)$



=10	$x=sqrt((e^{c})+(1^{2}))$	NotOccurIn15	NotOccurIn(x,NIL)
=8	$x^{2}=(e^{c})+(1^{2})$	Not10	¬Eq(x,NIL)
=6	$(x^{2})-(1^{2})=e^{c}$	Atom11	Atom(NIL)
=4	$(x+1)*(x-1)=e^{c}$	NotOccurIn16	NotOccurIn(x,2)
=1	ln((x+1)*(x-1))=c	Not9	¬Eq(x,2)
=3	ln(x+1)+ln(x-1)=c	Atom10	Atom(2)
NotOccurIn2	NotOccurIn(x,e)	NotOccurIn17	NotOccurIn(x,1)
Not3	¬Eq(x,e)	Not8	¬Eq(x,1)
Atom4	Atom(e)	Atom9	Atom(1)
OccursIn8	OccursIn(x,x-1)	NotOccurIn18	NotOccurIn(x,↑)
OccursIn10	OccursIn(x,(x 1))	Not7	¬Eq(x,↑)
OccursIn12	OccursIn(x,x)	Atom8	Atom(↑)
OccursIn3	OccursIn(x,x+1)	OccursIn20	$OccursIn(x,x^{2})$
OccursIn5	OccursIn(x,(x 1))	OccursIn22	OccursIn(x,(x 2))
OccursIn7	OccursIn(x,x)	NotOccurIn20	$NotOccurIn(x,(e^{c})+(1^{2}))$
NotOccurIn4	NotOccurIn(x,c)	NotOccurIn23	$NotOccurIn(x,(e^{c}\ (1^{2})))$
Not5	¬Eq(x,c)	NotOccurIn26	$NotOccurIn(x,((1^{2})))$
Atom6	Atom(c)	NotOccurIn29	$NotOccurIn(x,e^{c})$
OccursIn14	OccursIn(x,(x+1)*(x-1))	NotOccurIn32	NotOccurIn(x,(e c))
OccursIn16	OccursIn(x,((x+1)(x-1)))	NotOccurIn35	NotOccurIn(x,(c))
NotOccurIn6	$NotOccurIn(x,1^{2})$	NotOccurIn38	NotOccurIn(x,+)
NotOccurIn9	NotOccurIn(x,(1 2))	Not15	¬Eq(x,+)
NotOccurIn12	NotOccurIn(x,(2))	Atom16	Atom(+)

Figure 5.7: Equation Solving Example – Specific Explanation



=10	$?y8=\mathrm{sqrt}((?b1^{?y2})+(?v3^{2}))$	NotOccurIn9	NotOccurIn(x,(?v3 2))
=8	$?y8^{2}=(?b1^{?y2})+(?v3^{2})$	NotOccurIn12	NotOccurIn(x,(2))
=6	$(?y8^{2})-(?v3^{2})=?b1^{?y2}$	NotOccurIn41	NotOccurIn(x,NIL)
=4	$(?y8+?v3)*(?y8-?v3)=?b1^{?y2}$	NotOccurIn13	NotOccurIn(x,2)
=1	$\log_{?b1}((?y8+?v3)*(?y8-?v3))=?y2$	NotOccurIn10	NotOccurIn(x,?v3)
=2	$\log_{?b1}(?y8+?v3)+\log_{?b1}(?y8-?v3)=?y2$	NotOccurIn42	NotOccurIn(x,↑)
NotOccurIn39	NotOccurIn(x,?b1)	OccursIn20	OccursIn(x,$?y8^{2}$)
OccursIn8	OccursIn(x,?y8-?v3)	OccursIn22	OccursIn(x,(?y8 2))
OccursIn10	OccursIn(x,(?y8 ?v3))	NotOccurIn20	NotOccurIn(x,$(?b1^{?y2})+(?v3^{2})$)
OccursIn25	OccursIn(x,?y8)	NotOccurIn23	NotOccurIn(x,$((?b1^{?y2})(?v3^{2}))$)
OccursIn3	OccursIn(x,?y8+?v3)	NotOccurIn26	NotOccurIn(x,$(?v3^{2})$)
OccursIn5	OccursIn(x,(?y8 ?v3))	NotOccurIn29	NotOccurIn(x,$?b1^{?y2}$)
NotOccurIn40	NotOccurIn(x,?y2)	NotOccurIn32	NotOccurIn(x,(?b1 ?y2))
OccursIn14	OccursIn(x,(?y8+?v3)*(?y8-?v3))	NotOccurIn35	NotOccurIn(x,(?y2))
OccursIn16	OccursIn(x,((?y8+?v3)(?y8-?v3)))	NotOccurIn21	NotOccurIn(x,+)
NotOccurIn6	NotOccurIn(x,$?v3^{2}$)		

Figure 5.8: Equation Solving Example -- Generalized Explanation

CHAPTER 6

EGGS: PLAN EXPLANATIONS

Although constructing and verifying plans can be accomplished using logical deduction as described in section 4.2.6, other representations and algorithms are generally more concise and efficient [Chapman87, Fikes72, Sacerdoti74, Sacerdoti77, Waldinger77]. For example, verifying the following simple STRIPS plan took ten times as much CPU time using the EGGS deductive retriever compared to a more specialized algorithm (discussed later in this section) for verifying plans using STRIPS operators (44s versus 4.4s).

Given: InRoom(Robot,Room1), InRoom(Box,Room3), Connects(Door1,Room1,Room2)
Connects(Door2,Room2,Room3)

Goal: InRoom(Box,Room2)

Plan: GoThru(Robot,Door1,Room1,Room2), GoThru(Robot,Door2,Room2,Room3),
PushThru(Robot,Box,Door2,Room2,Room3)

In plan-based domains, it is generally more efficient to use representations especially developed for representing actions. This section discusses the EGGS system's ability to understand, generalize, and learn macro-operators from plan-based explanations and presents a couple of detailed examples of these processes.

6.1. Plans as Explanations: The STRIPS Example

The representation of actions used in EGGS for plan-based explanations is a slight variation of STRIPS operators [Fikes72]. Actions have *preconditions*, facts that must be true in order to perform the action, and *effects*, facts that are true after the action is performed. Both *add* and *delete* items are included in the effects; delete items simply have an explicit negation. Existing facts that are not explicitly negated by the effects of an action continue to be true. Definitions of the actions used in the STRIPS example are shown in Table 6.1. The departure from standard STRIPS operators which use separate add and delete lists allows a distinction to be made between facts that are known to be false and facts that are simply not known to be true. This is because facts that are known to false (i.e. deleted facts) are explicitly asserted as negations in the database. This in turn allows for negative preconditions that can check if a fact is known to be false, and allows positive effects to directly delete negative facts in the database.

Table 6.1: STRIPS Actions		
Action	Preconditions	Effects
GoThru(?a,?d,?r1,?r2)	InRoom(?a,?r1) Connects(?d,?r1,?r2)	InRoom(?a,?r2) ¬InRoom(?a,?r1)
PushThru(?a,?o,?d,?r1,?r2)	InRoom(?a,?r1) InRoom(?o,?r1) Connects(?d,?r1,?r2)	InRoom(?a,?r2) InRoom(?o,?r2) ¬InRoom(?a,?r1) ¬InRoom(?b,?r1)

Individual actions and Horn-clause rules form the *units* of an explanation and explanations are connected sets of actions and rules in which preconditions and antecedents are equated to effects and consequents. The specific explanation, explanation structure, and generalized explanation for the STRIPS example discussed earlier are shown in Figure 6.1, Figure 6.2, and Figure 6.3 respectively. In addition to action definitions given in Table 6.1, this example makes use of the following inference rule: Connects(?d,?r1,?r2) → Connects(?d,?r2,?r1).

6.2. Facilities in EGGS Supporting Plan Explanations

Figure 6.4 illustrates the the components in EGGS that use and learn plans. The complete EGGS system is equipped with a general purpose sub-system for verifying or "understanding" plans and a technique for obtaining general macro-operator definitions from generalized explanations. The current system does not have a planner for independently generating its own plans;

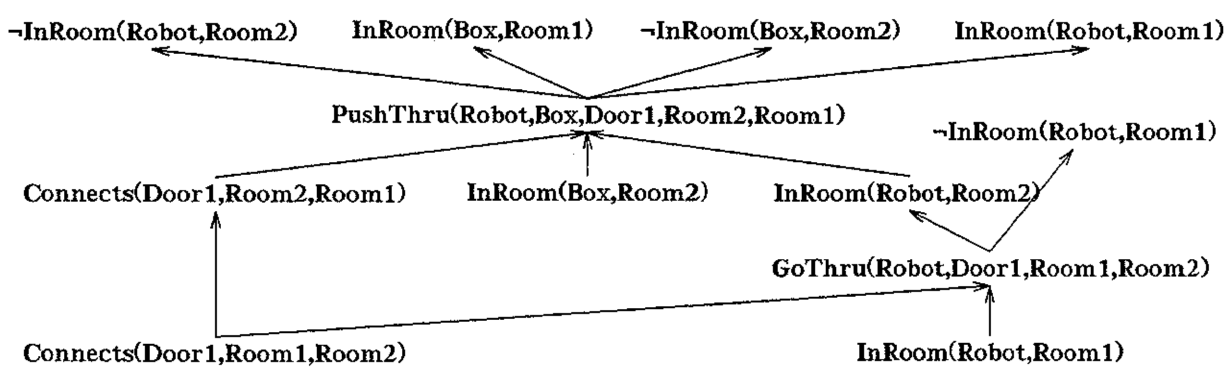


Figure 6.1: STRIPS Example -- Specific Explanation

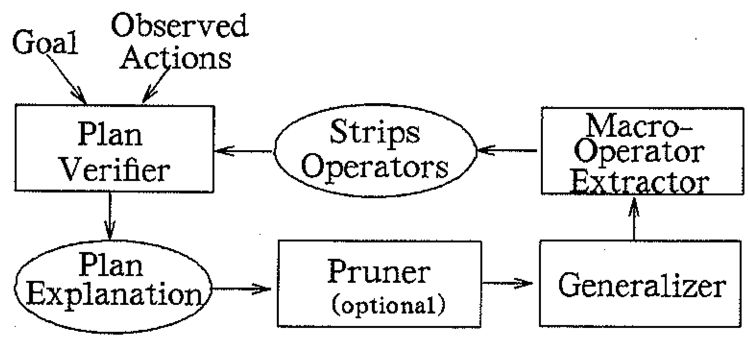


Figure 6.4: EGGS Architecture for Plan Explanations

however, there are many well-known planning algorithms which could be used for this task [Chapman87, Fikes72, Sacerdoti74, Sacerdoti77, Waldinger77].

6.2.1. Plan Verification

Verifying a plan is the task of constructing a complete explanation for a sequence of actions. This process may also be referred to as "explaining" or "understanding" the plan. An example of this task is generating the explanation shown in Figure 6.1 given only the following actions:

GoThru(Robot,Door1,Room1,Room2), PushThru(Robot,Box,Door1,Room2,Room1)

In addition to a set of actions performed, facts describing the initial state of the world and a fact describing the desired state of the world (the goal) may also be provided. The verification process attempts to causally connect these individual actions and states into a explanation that supports the goal (if provided). For the STRIPS blocks world, the process resembles understanding a simple "story" about a robot's actions.

The basic algorithm used in EGGS to explain simple robot plans is outlined in Figure 6.5. The function *Retrieve(x)* uses the EGGS deductive retriever (described in section 4.1.1) to try to infer the wff x. If successful, Retrieve asserts the retrieved fact in the database along with its dependencies and returns the retrieved fact. The function *Assert* adds a fact in the database and the function *Delete* removes a fact from the database.

The explanation procedure is given a chronologically ordered list of facts representing states and actions. If an input is a state, such as InRoom(Box,Room1), and it cannot be inferred from

```
for x in the list of input wffs do
    if x is a state and ¬Retrieve(x)
        then Assert(x)
    if x is an action
        then
            Assert(x)
            for each precondition p of x do
                let s = Retrieve(p)
                if s
                    then equate p to s
                    else Assert(p)
            for each effect e of x do
                Assert(e)
                let d = Retrieve(¬e)
                if d then Delete(d)
    if goal is known and Retrieve(goal)
        then generalize explanation supporting goal
```

Figure 6.5: Basic Plan Verification Algorithm

what is already known, then it is simply asserted in the database. If an input is an action, then the procedure first tries to infer that its preconditions are met. If a precondition cannot be inferred, then, since the action is known to have taken place, it is assumed to be true and asserted in the database. Next, the effects of the input action are asserted and any existing facts that directly contradict them are deleted from the database. After every input, if a goal has been specified, then the system tries to infer that the goal has been met. When the goal has been achieved, the resulting explanation is generalized.

As an example of this process, consider the standard STRIPS example mentioned above. In addition to the two actions, assume the system is given that the goal is InRoom(Box, Room1). Given the input GoThru(Robot, Door1, Room1, Room2), the preconditions InRoom(Robot, Room1) and Connects(Door1, Room1, Room2) cannot be inferred but can be consistently assumed, so they are asserted to be true. The effects of the GoThru: InRoom(Robot, Room2) and ¬InRoom(Robot, Room1), are then asserted. The second effect causes InRoom(Robot, Room1) to be deleted. The system then tries to infer that the goal, InRoom(Box, Room1), is satisfied and fails. Next, the input PushThru(Robot, Box, Room2, Room1) is considered. The precondition InRoom(Robot, Room2) was already asserted as an effect of the Gothru and the precondition Connects(Door1, Room2, Room1) can be inferred from Connects(Door1, Room1, Room2). The precondition InRoom(Box, Room2) is assumed as a necessary precondition for the action. The effects, InRoom(Robot, Room1), InRoom(Box, Room2), ¬InRoom(Robot, Room2), and ¬InRoom(Box, Room2) are asserted and the existing InRoom facts are deleted. The goal is then checked again and this time is successfully retrieved. The resulting explanation, shown in Figure 6.1, is then generalized to produce the generalization shown in Figure 6.3.

The basic understanding process presented here will only work on relatively simple plans in which every action is explicitly mentioned. Later, in chapter 10, this procedure is elaborated in order to allow it to construct explanations for natural language narratives. Nevertheless, the simple procedure presented here forms an important part of the understanding system in GENESIS.

6.2.2. Learning Macro-Operators

In order to be useful for later planning, the generalized explanation must be converted into a definition for a new action (a macro-operator) which summarizes the preconditions and effects of the combination of actions. The macro-operator learned from the STRIPS example is shown in Table 6.2. STRIPS used triangle tables to efficiently store macro-operators for all sub-sequences of the original plan. Currently, EGGS only builds macro-operators for complete sets of actions. However, unlike STRIPS, EGGS addresses the case in which the actions in the plan are only partially ordered instead of completely ordered. The additional goal is to find the minimally constrained partial ordering of the actions that allows them to be connected as they were in the example (i.e. maintaining the unifications between preconditions and effects).

Compared to a macro-rule learned by applying EGGS to a deductive formulation of planning (as in section 4.2.6), a macro-operator can be more informative and more general. A macro-rule like that presented in section 4.2.6 specifies only one effect of a combination of actions while a macro-operator specifies all of the effects. In addition, a macro-rule for a proof in situation calculus cannot represent partially ordered actions. Re-ordering actions in such a formulation requires completely altering the structure of the existing explanation. Consequently, generalizing the order of actions in such a formulation would be much more difficult.

Building a macro-operator requires determining the overall preconditions and effects of the general plan. The leaves of the generalized explanation form a set of preconditions for the macro-operator since they represent preconditions of actions in the plan that are not achieved by previous actions. For the STRIPS example, the first four preconditions from Table 6.2 are the leaves of the

Table 6.2: Macro-Operator Learned from the STRIPS Example	
GoThruPushThru(?a2,?d1,?x9,?b1,?d2,?x12,?y12)	
Preconditions	Effects
InRoom(?a2,?x9) InRoom(?b1,?x12) Connects(?d1,?x9,?x12) Connects(?d2,?y12,?x12) $\neg$(?a2=?b1 $\wedge$?x9=?x12)	InRoom(?b1,?y12) InRoom(?a2,?y12) $\neg$InRoom(?a2,?x12) $\neg$InRoom(?b1,?x12) ?y12$\neq$?x9 $\rightarrow$ $\neg$InRoom(?a2,?x9)

generalized explanation shown in Figure 6.3. The final precondition in the table is added later and is discussed in the following section.

The effects of the overall plan are more difficult to determine since, as noted in [Fikes72 (section 4.3)], the deletions of facts that occur in general may not be the same as the deletions that occurred in the specific instance. To illustrate this point, they use the following simple plan for pushing two boxes to two separate locations: Push(Box1, Loc1), Push(Box2, Loc2). The initial generalization for this plan is: Push(?b1, ?l1), Push(?b2, ?l2) which has the effects: At(?b1, ?l1), At(?b2, ?l2). However, if ?b1 and ?b2 are instantiated to the same box and ?l1 and ?l2 are instantiated to distinct locations, this plan miraculously achieves the goal of having the same box be in two places at once. The problem is that in certain situations the actions in a macro-operator may delete facts that were not deleted in the specific example. Consequently, in order to determine which facts are still true at the end of the execution of the macro-operator, one must determine all deletions that might occur in the general case.

6.2.2.1. Determining Deletions and Preventing Protection Violations

Determining possible deletions is discussed in [Fikes72 (section 4.3)]; however, the problem is more complicated when one considers partially ordered actions. Most ordering constraints in a plan are imposed by one action achieving a precondition for another action. These orderings are captured by the explanation structure: an action must precede another action if it eventually supports it. Any topologically sorted [Reingold77, Sedgewick83] list of the actions in the generalized explanation graph will be an ordering that satisfies these constraints. However, some of these orderings may result in *protection violations* [Charniak85, Sussman73] in which an action deletes or *clobbers* a goal or a precondition for a later action. When examining deletions in a macro-operator, it is important to notice possible protection violations and impose additional ordering constraints to avoid them.

Figure 6.6 describes the algorithm EGGS uses to determine deletions for the generalized explanation and impose additional ordering constraints to prevent protection violations. In the algorithm, *Supports?(a, b)* refers to a function that returns true iff a eventually supports b in the generalized explanation, i.e. if there is a directed path from a to b in the explanation graph. For each action, the algorithm determines the set of states in the explanation that could be true before the action is executed based only on the partial ordering of actions imposed by the explanation

```
for each action a in the generalized explanation do
   let deletable = Ø
   for each state s in the generalized explanation do
      if ¬Supports?(a, s) then let deletable = deletable ∪ {s}
   for each effect e of a do
      for each state d in deletable do
         if ¬d unifies with e
            then
               let α be the conditions under which ¬d and e unify
               let c be the action of which d is an effect (possibly NIL)
               if null(c) ∨ Supports?(c, a)
                  then
                     if d is the goal or supports a proof of the goal
                        then add ¬α to the preconditions
                        else d is deleted by e when α is true
                  else
                     if d is the goal or supports a proof of the goal
                        then a must precede c when α is true
                        else d is deleted by e when α is true and c precedes a
               PreventPreconditionClobbering(a, d, c, α)

procedure PreventPreconditionClobbering(a, d, c, α)
(* Add conditions to prevent a from clobbering precondition d when α is true *)
   for each action b such that d is a precondition of b or supports a proof of a precondition of b do
      if Supports?(a, b)
         then
            if null(c) ∨ Supports?(c, a)
               then add ¬α to the preconditions
               else a must precede c when α is true
         else
            if null(c) ∨ Supports?(c, a)
               then b must precede a when α is true
               else a must precede c or b must precede a when α is true
```

Figure 6.6: Algorithm for Determining Deletions and Preventing Protection Violations

structure. These are called the *deletable* states since they are ones that the action could possibly delete. For each of these states, the condition (α) under which the action could delete the state in general are determined. For example, if an effect of an action is ¬InRoom(?x, ?y), then the state InRoom(?a, ?b) is deleted iff ?x=?a and ?y=?b. If the deletable state is not supported by another action (i.e. it is a precondition of the macro-operator) or if it is an effect of an action that must precede the deleting action because it supports it, then the state must be true before the action is executed and it will be deleted when α is true. If the deletable state is also the goal or supports a proof of the goal, then $\neg\alpha$ is added to the preconditions to prevent the goal from being clobbered. Otherwise it is marked as being deleted when α is true. If the deletable state is not necessarily true before the action is executed, then it will be deleted when both α is true and the action supporting it precedes the deleting action. If the deletable state is the goal or supports a proof of the goal, then in order to prevent the goal from being clobbered, the deleting action must precede the action supporting the deleted state when α is true.

The procedure PreventPreconditionClobbering adds additional ordering constraints and preconditions to the macro-operator in order to prevent a precondition from being clobbered when action a deletes state d. If an effect of action a deletes a precondition p of action b, then in order to avoid a protection violation either b must precede a or a must precede action c where p is an effect of c [Sacerdoti77, Sussman73]. However, the existing partial ordering of the actions imposed by the explanation structure may rule out either or both of these possibilities. If both possibilities are ruled out, then $\neg\alpha$ is added to the list of preconditions for the macro-operator to prevent the plan from being used in such situations. Otherwise, the possible ordering constraints that will prevent the protection violation are recorded.

As one example of the process of determining deletions and preventing protection violations, consider the standard STRIPS example. Below is the trace produced by the system when examining the deletions for the generalized explanation shown in Figure 6.3.

Plan not valid when ?a2=?b1 ∧ ?x9=?x12 because ¬InRoom(?a2,?x9) clobbers InRoom(?b1,?x12)
In general ¬InRoom(?a2,?x9) deletes InRoom(?a2,?x9) when T
In general InRoom(?a2,?y12) deletes ¬InRoom(?a2,?x9) when ?y12=?x9
In general InRoom(?b1,?y12) deletes ¬InRoom(?a2,?x9) when ?b1=?a2 ∧ ?y12=?x9
In general ¬InRoom(?b1,?x12) deletes InRoom(?b1,?x12) when T
In general ¬InRoom(?b1,?x12) deletes InRoom(?a2,?x9) when ?b1=?a2 ∧ ?x12=?x9
In general ¬InRoom(?b1,?x12) deletes InRoom(?a2,?x12) when ?b1=?a2

In general ¬InRoom(?a2,?x12) deletes InRoom(?b1,?x12) when ?a2=?b1

In general ¬InRoom(?a2,?x12) deletes InRoom(?a2,?x9) when ?x12=?x9

In general ¬InRoom(?a2,?x12) deletes InRoom(?a2,?x12) when T

One important thing to notice is that ¬(?a2=?b1 ∧ ?x9=?x12) is added to the list of preconditions for the plan in order to prevent the GoThru action from clobbering a precondition of the PushThru. Reordering the steps is ruled out since the explanation structure requires that the GoThru precede the PushThru and since the clobbered precondition is a precondition of the entire macro-operator and consequently is not achieved by an action that can be performed after the GoThru. Actually, for ?a2 to be equal to ?b1 would require that an agent be able to push himself through a door, and for ?x9 to be equal to ?x12 would require that an agent be able to go through a door from a room back to the same room. In a better formulation of the domain, both of these conditions could be disallowed by adding additional preconditions to GoThru and PushThru.

6.2.2.2. Complexity of Determining Deletions and Preventing Protection Violations

The worst-case time complexity of the algorithm in Figure 6.6 is relatively easy to determine. In the following discussion, loops will be referred to by their iterative variables (e.g. the a loop, the e loop, etc.).

Let the number of nodes in the generalized explanation graph be n. Since the function *Supports?* determines whether or not there is a directed path from one node to another in this graph, in the worst case it must traverse the entire graph. Since traversing a graph is $O(|V|+|E|)$ where $|V|$ is the number of nodes and $|E|$ is the number of edges and since $|E| \leqslant |V|^2$, the worst case complexity of a call to *Supports?* is $O(n^2)$.

Since the number of actions must be less than n, the outermost loop (the a loop) is executed at most n times. Since the number of states is also less than n, the body of the loop for determining the set of deletable states (the s loop) is executed at most n times for each action for a maximum total of n^2 times. Since the body of this loop includes a call to *Supports?* which is $O(n^2)$, the total maximum time needed for all executions of the s loop is $O(n^4)$. Since each deletable set is a subset of the set of states, its maximum cardinality is n.

Since the total number of effects of actions must be less than the number of states, the body of the e loop is executed at most a total of n times. Since n is the maximum size of a deletable set, the body of the d loop is executed at most n times for each effect for a maximum total of n^2 times.

In addition to the call to PreventPreconditionClobbering, the operations in the body of the d loop that depend on the size of n include a call to *Supports?* and determining whether a state supports a proof of the goal. Since both of these operations involve finding a path between two nodes in the explanation graph, they take at most $O(n^2)$ time. Therefore, let the total maximum time needed for all executions of the e loop be:

$$n^2(O(n^2) + P)$$

where P is the time complexity of a call to PreventPreconditionClobbering.

The body of the b loop in PreventPreconditionClobbering is executed less than n times since the *total* number of actions is less than n. Since executing the body of the loop requires two calls to *Supports?* which takes $O(n^2)$ time, the maximum time required for a call to this function is $O(n^3)$. Therefore, the total maximum time needed for all executions of the e loop is:

$$n^2(O(n^2) + O(n^3)) = O(n^5)$$

Since, the total maximum time needed for all executions of the s loop was calculated to be only $O(n^4)$, the worst case time complexity of the complete process is also $O(n^5)$.

The fact that the $O(n^2)$ *Supports?* function is called within bodies of loops that are executed $O(n^2)$ and $O(n^3)$ times is a dominant factor in the complexity of the complete algorithm. If a call to this function could be reduced to constant time, a minor modification of the preceding analysis demonstrates that the complexity of the complete process would then be $O(n^3)$. The maximum time required for all executions of the s loop becomes $O(n^2)$. The total maximum time for all executions of the e loop becomes:

$$n^2(c + P)$$

where c is a constant. The time complexity of a call to PreventPreconditionClobbering (P) becomes $O(n)$ and the overall worst case complexity becomes:

$$O(n^2) + n^2(c + O(n)) = O(n^3)$$

Since *Supports?(a,b)* simply determines whether or not there is a directed path from a to b, the value of this function for all pairs of the n nodes in the explanation graph can be determined by computing the graph's *transitive closure*.[1] Since there exist $O(|V|^3)$ algorithms for computing the transitive closure of a directed graph [Reingold77], all the values of *Support?* for a particular explanation can be pre-computed in $O(n^3)$ time. Calls to this function will then take constant time and

[1]The *transitive closure* of a directed graph G=(V,E) is a directed graph G′=(V,E′) such that there is an edge $u \rightarrow v$ in E′

the complete process of determining deletions and protection violations can be accomplished in $O(n^3)$ time.

In conclusion, although the algorithm implemented in EGGS is $O(n^5)$, computation of the the transitive closure of the generalized explanation graph using one of the known $O(|V|^3)$ algorithms would result in an $O(n^3)$ algorithm for the problem of determining deletions and preventing protection violations.

6.2.2.3. Determining the Effects of a Macro-Operator

Once all possible deletions have been marked, determining the effects of the over-all macro-operator is quite simple. The effects of each individual action are considered in turn. If an effect was never marked as being deleted under any condition, then it is added to the list of effects for the macro-operator. In the STRIPS example, this accounts for the first four effects shown in Table 6.2. If a state was marked as being deleted under some set of conditions, then the overall effect is given by the following implication: if none of the deletion conditions are met, then the state is true. In the STRIPS example, the state ¬InRoom(?a2, ?x9) was marked as deleted if and only if ?y12=?x9. Consequently, the final effect of the macro-operator is: ¬(?y12=?x9) → ¬InRoom(?a2, ?x9). This simply states that unless the agent pushes the object back to the room he originally started in (as in the observed instance), then he is no longer in the starting room.

6.2.2.4. Detecting Inconsistent Ordering Constraints

One of the final steps in building a macro-operator is the detection and prevention of inconsistent ordering constraints. The ordering constraints posted to prevent protection violations may contradict each other in certain situations. For example, one constraint may state that action A must proceed action B when α is true while another states that B must proceed A when β is true. In order to detect such situations, a resolution theorem prover is used to determine sets of ordering constraints that are inconsistent.[2] In addition to the posted ordering constraints, the theorem prover is given the following two axioms:

Before(?a,?b) ∧ Before(?b,?c) → Before(?a,?c)

Before(?a,?b) → ¬Before(?b,?a)

---

if and only if there is a (non-empty) directed path from u to v in E [Even79].

[2] Because of disjunctive ordering constraints, a simple cycle-detecting algorithm cannot be used to find all sets of inconsistent constraints.

When the resolution theorem prover finds a contradiction, it returns the set of axioms upon which the proof depends. In order to prevent such a contradiction from arising during the subsequent use of the macro-operator, an additional precondition must be added. Assuming each of the n ordering constraints c_i in the contradictory set must be satisfied when condition α_i is true, the following precondition must be added to the macro-operator: $\neg(\alpha_1 \wedge \alpha_2 \ldots \wedge \alpha_n)$ In the example above, the wff: $\neg(\alpha \wedge \beta)$ would be the necessary additional precondition. A more concrete example is given in section 6.3.1.

Of course, using a resolution theorem prover to find all sets of contradictory axioms is a computationally intractable process which is not even guaranteed to halt. In EGGS, the theorem prover is given a time limit, after which it stops and returns proofs for all the contradictions it has found so far. Consequently, the system does not actually guarantee the prevention of ordering contradictions during later instantiation. Should such a contradiction arise, one could perform explanation-based learning from failure to modify the plan at that time in order to prevent similar problems in the future. Some actual systems that use EBL to learn from failures due to unforeseen interactions are described in [Gupta87] and [Chien87a].

6.2.2.5. Simplifying the Results

The final set of wffs describing the preconditions, effects, and ordering constraints for a macro-operator can be unnecessarily complex. Consequently, EGGS uses a system for simplifying logical expressions to clean up the final results. As described in [Minton87a], simplification or *compression* helps to decrease the matching cost needed to determine whether or not a macro-operator applies.

The simplifier currently used by EGGS first converts expressions to conjunctive normal form and eliminates tautological and subsumed clauses. Next, a set of axioms supplied by the user for each domain are used to further simplify the CNF formula. If a literal l in a clause together with the domain axioms and other clauses implies another literal in the clause, then the literal l is removed. The EGGS deductive retriever is used to prove the implication. This is a valid simplification since:

$$((P \wedge R) \rightarrow Q) \rightarrow ((R \wedge (P \vee Q)) \equiv (R \wedge Q))$$

Therefore, if $(P \wedge R) \rightarrow Q$ is true in the domain, then the reduced formula is equivalent to the original. Clauses that are implied by other clauses together with the domain axioms are also removed. Once again, the EGGS deductive retriever is used to prove the implication. This simplification is valid since:

$$(P \rightarrow Q) \rightarrow ((P \wedge Q) \equiv P)$$

Therefore, if $P \rightarrow Q$ is true in the domain, then the reduced formula is equivalent to the original. Finally, DeMorgan's law is used to convert clauses that are more compactly represented as negated terms. That is, clauses of the form:

$$\neg l_1 \vee \neg l_2 \ldots \vee \neg l_{n-1} \vee \neg l_n$$

are changed to:

$$\neg(l_1 \wedge l_2 \ldots \wedge l_{n-1} \wedge l_n)$$

A concrete example of the amount of simplification possible using this method is given in section 6.3.1. Of course, the EGGS simplifier is not guaranteed to produce optimal simplifications; however, general problems of minimizing logical expressions are known to be NP-Complete (see Minimum Axiom Set and Minimum Disjunctive Normal Form in [Garey79]).

6.2.2.6. Relation to Nonlinear Planning

Work in *nonlinear planning* [Sacerdoti77, Tate76] has addressed the issue of building plans with partially ordered actions by initially assuming there are no goal interactions and then detecting protection violations and imposing ordering constraints to prevent them. Although the task of building a specific partially-ordered plan for achieving a specific goal is quite different from the task of generalizing a specific totally-ordered plan into a general partially-ordered macro-operator, some of the underlying processes are quite similar.

In [Nilsson80], the procedure for detecting and preventing protection violations in nonlinear planning is called DCOMP. The procedure in Figure 6.6 is in some ways more general and in some ways less general than DCOMP. It is more general because it must determine the conditions (α) under which a deletion will take place in the generalized plan. DCOMP works with a specific instance of a plan and therefore does not need to determine which instantiations of a plan result in a deletion. The procedure in Figure *6.6 is less general than DCOMP because it does not consider alternative ways of achieving the preconditions of an action. The explanation structure explanation determines how preconditions were met in the particular example, and these constraints are retained in the generalization. Using the terminology from [Nilsson80], the procedure presented here does not determine all of the possible *adders* of a precondition in the plan since each precondition is assumed to added by the action that added it in the original specific instance. Of course, an even more general macro-operator could be produced by generalizing the explanation structure and considering alternative adders. However, this approach would even further complicate the procedure for generating a macro-operator.

The actual process of determining what Nilsson calls a *noninteractive* ordering for a particular instantiation of a plan must be done when the final macro-operator is used to solve a future problem. Assuming all inconsistent sets of ordering constraints were detected and prevented by adding additional preconditions to the macro-operator, for any situation that meets the preconditions, there is guaranteed to be a partial ordering of the actions that satisfies all of the ordering constraints. However, due to the possible presence of disjunctive ordering constraints, actually finding it may require an expensive search. However, this should not be surprising since the increased generality of a partially-ordered macro-operator can clearly result in a corresponding decrease in operationality.

6.3. Plan Examples

Unfortunately, the STRIPS example does not illustrate many of the aspects involved in constructing macro-operators with partially ordered actions because the explanation structure itself imposes a total ordering on the two actions in the plan. This section presents an additional example involving program assignment statements in which the actions are not totally ordered. In addition to STRIPS and variable assignment examples like the ones presented in this chapter, the system has also been tested on classical block stacking examples (like those in [Sussman73]) and narrative understanding examples (see chapters 9-11 and appendix B, in particular the Arson example in section B.2 contains an interesting potential protection violation).

6.3.1. Variable Assignment Example

An action definition for assigning a value to a variable modelled after LISP's SETQ function is shown in Table 6.3. Setq(?a,?b) sets variable ?a to the value of variable ?b and Value(?a,?x) means that variable ?a has value ?x. To make the problem interesting, it is assumed that we are modelling parallel computation in which any number of Setq's can be performed simultaneously. As a result, determining possible protection violations in this domain is equivalent to detecting interactions between parallel computations; and imposing ordering constraints is equivalent to forcing certain statements to be executed sequentially.

Consider the following simple problem and the plan for solving it:

Given: Value(A,B), Value(X,Y), Value(R,N), Value(S,M)

Table 6.3: Variable Assignment Actions		
Action	Preconditions	Effects
Setq(?a,?b)	Value(?a,?x) Value(?b,?y)	Value(?a,?y) ¬Value(?a,?x)

Goal: Value(A,N) ∧ Value(X,M)

Plan: Setq(A,R), Setq(X,S)

The specific and generalized explanations constructed by EGGS while verifying this plan are shown in Figure 6.7 and Figure 6.8 respectively. Notice that the explanation structure does not impose an ordering on the two Setq operators. A trace of the process for determining deletions and preventing protection violations for this example is given below:

In general Value(?a2,?y2) deletes ¬Value(?a1,?x1) when (?a2=?a1 ∧ ?y2=?x1) and Setq(?a1,?b1) precedes Setq(?a2,?b2)

In general ¬Value(?a2,?x2) deletes Value(?b2,?y2) when (?a2=?b2 ∧ ?x2=?y2)

In general ¬Value(?a2,?x2) deletes Value(?a2,?x2) when T

Setq(?a1,?b1) should precede Setq(?a2,?b2) when (?a2=?b1 ∧ ?x2=?y1) or else ¬Value(?a2,?x2) will clobber Value(?b1,?y1)

In general ¬Value(?a2,?x2) deletes Value(?b1,?y1) when (?a2=?b1 ∧ ?x2=?y1)

Setq(?a1,?b1) should precede Setq(?a2,?b2) when (?a2=?a1 ∧ ?x2=?x1) or else ¬Value(?a2,?x2) will clobber Value(?a1,?x1)

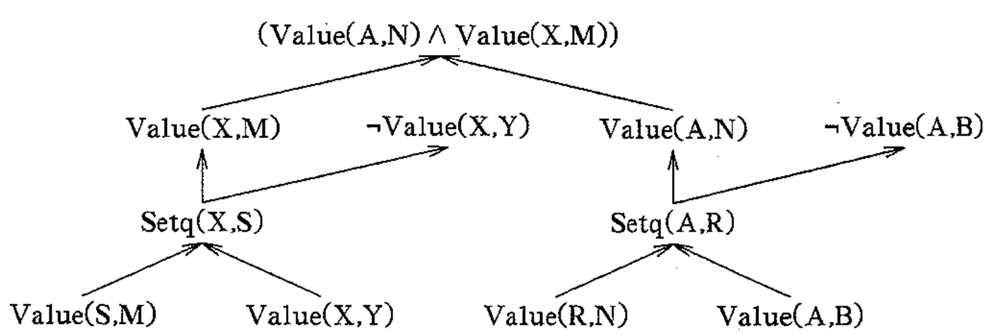


Figure 6.7: Variable Assignment Example -- Specific Explanation

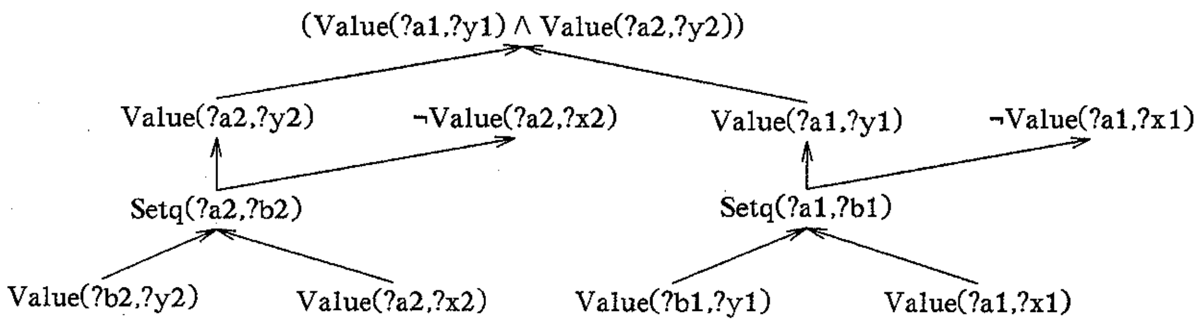


Figure 6.8: Variable Assignment Example -- Generalized Explanation

In general ¬Value(?a2,?x2) deletes Value(?a1,?x1) when (?a2=?a1 ∧ ?x2=?x1)

Setq(?a2,?b2) should precede Setq(?a1,?b1) when (?a2=?a1 ∧ ?x2=?y1) or else ¬Value(?a2,?x2) will clobber goal: Value(?a1,?y1)

In general Value(?a1,?y1) deletes ¬Value(?a2,?x2) when (?a1=?a2 ∧ ?y1=?x2) and Setq(?a2,?b2) precedes Setq(?a1,?b1)

Setq(?a2,?b2) should precede Setq(?a1,?b1) when (?a1=?b2 ∧ ?x1=?y2) or else ¬Value(?a1,?x1) will clobber Value(?b2,?y2)

In general ¬Value(?a1,?x1) deletes Value(?b2,?y2) when (?a1=?b2 ∧ ?x1=?y2)

Setq(?a2,?b2) should precede Setq(?a1,?b1) when (?a1=?a2 ∧ ?x1=?x2) or else ¬Value(?a1,?x1) will clobber Value(?a2,?x2)

In general ¬Value(?a1,?x1) deletes Value(?a2,?x2) when (?a1=?a2 ∧ ?x1=?x2)

Setq(?a1,?b1) should precede Setq(?a2,?b2) when (?a1=?a2 ∧ ?x1=?y2) or else ¬Value(?a1,?x1) will clobber goal: Value(?a2,?y2)

In general ¬Value(?a1,?x1) deletes Value(?b1,?y1) when (?a1=?b1 ∧ ?x1=?y1)

In general ¬Value(?a1,?x1) deletes Value(?a1,?x1) when T

(Before(Setq(?a2,?b2),Setq(?a1,?b1)) ∧ Before(Setq(?a1,?b1),Setq(?a2,?b2)))
results in a contradiction. Add precondition: ¬(((?a2=?a1 ∧ ?x2=?y1) ∨ (?a1=?b2 ∧ ?x1=?y2) ∨ (?a1=?a2 ∧ ?x1=?x2)) ∧ ((?a2=?b1 ∧ ?x2=?y1) ∨ (?a2=?a1 ∧ ?x2=?x1) ∨ (?a1=?a2 ∧ ?x1=?y2)))
which simplifies to ¬(?a1=?b2 ∧ ?a2=?b1) ∧ ¬(?a2=?a1)

The system detects that each Setq could potentially clobber a precondition of the other one if certain variables are the same. Ordering constraints are posted to prevent protection violations in these situations. However, these ordering constraints contradict each other in certain situations. For example, when ?a1=?a2 and ?x1=?x2 these constraints specify that each of the two Setqs should precede the other since in this situation each one clobbers a precondition of the other. The process described in section 6.2.2.4 detects such possible contradictions and adds preconditions to prevent them. In the example, this process results in both ¬(?a2=?a1) and ¬(?a1=?b2 ∧ ?a2=?b1) being added as preconditions to the macro-operator. When simplifying the final results, the system made use of the following domain axiom: Value(?x,?a) ∧ Value(?y,?b) ∧ ?x=?y → ?a=?b. This axiom simply states that a variable can have only one value at a time. Table 6.4 shows the macro-operator definition resulting from the complete packaging process.

In summary, the system notices that if the two variables being set (?a1 and ?a2) are the same, then the goal of having them set to two different values cannot be achieved. Also, if either assignment references the variable set by the other, then the two cannot be executed in parallel. In these cases, the assignments must be properly ordered to prevent the referenced variable from being reset before it is used. Finally, if each assignment references the variable set by the other, then two assignments cannot solve the problem. This is the classic "variable swapping" problem which requires the use of a temporary variable (see section 7.5.3 of [Nilsson80].).

Table 6.4: Macro-Operator Learned from the Variable Assignment Example

SetqSetq(?a1,?b1,?x2,?y2,?x1,?y1,?a2,?b2)		
Preconditions	Effects	Orderings
Value(?a1,?x1) Value(?a2,?x2) Value(?b1,?y1) Value(?b2,?y2) ?a2≠?a1 ¬(?a1=?b2 ∧ ?a2=?b1)	Value(?a1,?y1) Value(?a2,?y2) ¬Value(?a1,?x1) ¬Value(?a2,?x2)	?a1=?b2 → Before(Setq(?a2,?b2),Setq(?a1,?b1)) ?a2=?b1 → Before(Setq(?a1,?b1),Setq(?a2,?b2))

CHAPTER 7

EGGS: EMPIRICAL RESULTS ON THE EFFECT OF LEARNING

This chapter summarizes some empirical results on the effect of learning on problem solving performance in all of the domains discussed in chapters 3, 4 and 5[1]. The macro-rules learned by EGGS affect its future performance in various ways. After learning a macro-rule from a particular example, performance on similar problems is enhanced. However, performance may be degraded on problems that are superficially similar but can not be solved using the macro-rule.

7.1. Performance Improvement

Table 7.1 summarizes empirical results on *positive transfer*, the ability of learning to improve problem solving performance on similar problems. It gives the CPU time in seconds required to solve a problem both before and after learning a macro-rule capable of solving the problem in one step. In each of the domains, the learned rule is the one discussed in the associated section in chapter 3, 4, or 5. The CPU time required to learn the rule is also given, where learning includes pruning and generalizing the explanation as well as packaging it into a macro-rule. Performance was tested on solving the test problem discussed for each domain before and after learning the rule

Table 7.1: Speedup Due to Learning

Example	Learning CPU sec	Solution CPU sec Before	Solution CPU sec After	Speedup
CUP	1.08	4.82	3.70	1.30
Suicide	.76	1.69	1.38	1.22
SafeToStack	.81	3.69	2.72	1.36
LEAP	.73	3.99	.79	5.05
MA	.67	759.00	.53	1432.00
LEX2	.76	2.02	1.31	1.54
Geometry	1.26	5.71	1.53	3.73
Geometry (difficult)	5.52	>1680.00	4.02	>418.00
STRIPS (deductive)	6.11	14.50	3.94	3.68
LEAP (rewrite)	.35	1.10	.49	2.24
LEX2 (rewrite)	.46	1.60	1.09	1.47
Equation Solving	8.08	21.40	8.78	2.44

[1]Examples in chapter 6 are not considered since EGGS does not have a planner for independently solving problems using STRIPS operators.

from the different but related learning example for that domain.[2] In all of the theorem-proving examples, the backward-chaining depth bound was set to ten, the depth of the shortest known proof for the hardest problem (the "difficult geometry" example).

The *speedup* is the ratio of the two run times and represents how much faster the system solves the test problem after learning. In the simple examples where little or no search is involved, the speedup is only about 1.3; however, in the more realistic examples with larger search spaces, the speedup ranges from about 4 to 1400 times faster. Since the difficult geometry problem shown in Figure 4.17 could not even be solved before learning, the speedup due to learning from a sketchy proof can only be given a lower bound of about 400.

Empirical results on positive transfer resulting from explanation-based learning have also been reported for the PROLEARN learning PROLOG system [Prieditis87]. However, only one of the PROLEARN examples involves *any* search, and this is the simple SafeToStack example which requires minimal search. Therefore, no large speedups, like those reported here for the MA and difficult Geometry examples, are presented in the PROLEARN results. More comprehensive experiments in the single domain of proving theorems in elementary logic (the MA domain) are reported in [O'Rorke87a, O'Rorke87b]. These experiments demonstrated that explanation-based learning can improve performance to a greater degree than rote learning. On a set of 52 theorems from *Principia Mathematica* [Whitehead13], an EBL system using depth-bounded search proved 83% of the theorems, compared to 79% for a rote-learning system, and 31% for a non-learning system (all systems were given the same depth-bound). Over all the problems, the EBL system searched 32% *less* nodes than the non-learning system, while the rote learning system searched 20% *more* nodes than the non-learning system.[3] Some recent experiments that compare a non-learning system, a standard EBL system, and an EBL system that can learn iterative macro-operators are reported in [Shavlik88].

In order for the learning of macro-rules to contribute to the overall performance of a system, there must be regularity in the class of problems to which the system is applied. It must be assumed that it is likely that the system will encounter a problem that is similar to some past problem from which it has already acquired a macro-rule. Fortunately, this assumption is probably not an unreasonable one since most domains are characterized by classes of problems that can

[2]Modified versions of the learning examples were used to test the Cup and Suicide examples presented in chapter 3.

[3]These percentages were calculated from data presented in [O'Rorke87b, Table 3.1]

be solved using similar techniques. The advantage of learning from examples taken from the environment instead of randomly generating macro-rules for various combinations of rules is that the examples will hopefully characterize the types of problems the system will have to solve in the future.

7.2. Performance Degradation

In addition to performance gains, there is also a price to pay for learning macro-rules. When the system attempts to solve a problem that cannot be solved using the macro-rule, it wastes time unsuccessfully trying to apply the rule. The result is *negative transfer*, a decrease in problem solving performance on superficially similar problems after learning. Table 7.2 summarizes some empirical data that illustrates negative transfer. It gives the CPU time in seconds required to solve a problem before and after learning a rule that cannot be used to solve the problem. Once again, theorem-proving examples were run with a depth-bound of ten.

Slowdown is the ratio of the two run times, which represents how much faster the system solved the problem *before* learning. The test problems were constructed with the goal of maximizing this slowdown effect. This was done by attempting to construct goals that match the conclusion of the macro-rule, thereby causing the system to try (unsuccessfully) to use the rule to achieve the goal. Unfortunately, this was only possible for the SafeToStack, LEX2, Geometry, and STRIPS examples since the macro-rules for the LEAP and MA examples do not have antecedents and equation solving examples that met the criteria could not be solved without adding a large

Table 7.2: Slowdown Due to Learning

Example	Test Problem	Solution CPU sec		Slowdown
		Before	After	
SafeToStack	SafeToStack(Obj1,Obj2) (Given $\neg$Fragile(Obj2))	1.06	1.09	1.03
LEAP	Equiv($\neg(\neg a \vee b)$,?x)	1.49	1.50	1.01
MA	NIL $\Rightarrow$ P$\rightarrow$(P$\vee$Q)	21.10	21.90	1.04
LEX2	Solution($\int 3x^{-1}dx$,?a)	2.19	2.39	1.09
Geometry	$\Delta ABD \cong \Delta CBD$ (Given $\overline{AB} \cong \overline{CB}$, $\overline{AD} \cong \overline{CD}$)	4.63	8.18	1.77
STRIPS (deductive)	Achieve(inroom(Robot,Room2),?s) (Given facts for Figure 3.6)	3.62	>2100.00	>580.00
LEAP (rewrite)	$\neg(\neg a \vee b)$	.64	.69	1.08
LEX2 (rewrite)	$\int 3x^{-1}dx$	1.28	1.41	1.10
Equation Solving	ln(x+3)+ln(5)=c	11.40	11.70	1.03

number of extra rules. In an attempt to maximize slowdown in these remaining cases, problems were constructed which match the conclusion of the macro-rule as much as possible.

In these experiments, the slowdown effect of learning a particular rule was generally much less than the speedup, so the gain of learning the macro-rules probably out-weighs the loss. In the STRIPS example, however, learning causes the system to waste a great deal of time attempting to use the macro-rule on a relatively simple problem that is much more easily solved from first principles. The system eventually exhausts its storage before it abandons its attempt to use the new macro-rule to solve the problem. [Minton85] describes a more comprehensive experiment in a single domain very much like STRIPS' domain which demonstrates that indiscriminately learning macro-operators can eventually lead to the slowdown effect dominating and result in an overall decrease in performance.

In these cases, it is important to be selective about which macro-rules are created and which are retained. The problem of determining whether a macro-rule's positive contribution to performance outweighs its negative effect has been called the *utility problem* [Minton87a]. The possibility of generating large numbers of macro-operators which adversely affect performance was first mentioned in the conclusion of the STRIPS learning paper [Fikes72]. Fikes *et. al.* proposed that a possible solution to this problem might be the inclusion of a mechanism for forgetting infrequently used macro-operators. This mechanism would keep statistics on the frequency with which each macro-operator is used and discard those which fell below a certain threshold. The learning of *S-Macros* in MORRIS [Minton85] is an implementation of this proposal. Experiments conducted with MORRIS indicate that this technique can result in improved performance [Minton85]. Analytical methods for judging the *utility* of macro-operators (i.e. their ability to improve performance) have also been developed and tested [Iba85, Minton85, Minton87a]. If macro-operators that are known to prevent backtracking (*T-Macros*) are selectively retained, empirical results indicate that overall performance can be improved [Minton85]. Some additional untested suggestions for dealing with the problem of slowdown are given in [Prieditis87]. Unlike STRIPS or the current PROLEARN system, GENESIS does not learn from every explanation. The criteria GENESIS uses to determine when an explanation is worth learning from are discussed in chapter 11.

7.3. A Suggestion for Better Controlled Experiments

Macro-rules can improve a resource-bounded problem solving system in two important ways. It can increase the class of problems the system is inherently capable of solving and it can increase the speed of problem solving. A good experimental comparison of the performance of a learning and non-learning system must consider both of these factors. An experiment that examines only one of these factors without controlling for the other can produce misleading results.

Specifically, when the problem solver is depth-first and depth-bounded, like the theorem prover in EGGS and in [O'Rorke87b], performance data is actually biased against a system that learns macro-rules. Such a system can waste a great deal of time trying to use a macro-rule to solve a problem by combining it with other macro-rules and with rules in the original domain theory. With the same depth-bound for both the learning and non-learning systems, a system that learns macro-rules is capable of solving a larger set of problems. This is because a macro-rule that can represent an arbitrarily deep proof in the original domain theory counts as only one rule application when used to solve new problems. Therefore, the search space the learning system must explore is constantly getting larger since learned rules increase the branching factor while the depth of its search tree remains constant. On the other hand, the search space the non-learning system must explore remains constant.

For example, in the STRIPS example, since the learned macro-rule has depth 7, in some places the learning system is exploring the search space of the original domain theory to a depth of 16 instead of 10 when using this rule. This in turn accounts for the large amount of time spent trying to use this particular macro-rule on the slowdown-testing problem for the STRIPS domain.

In summary, in order for the overall execution time of the learning system to be less than the non-learning system's, it must be capable of both solving more problems *and* solving them faster. A fairer comparison of overall search or execution time would require a controlled experiment in which the class of problems solvable by both systems remained constant. This could be done by marking the depth of each antecedent of a macro-rule in terms of the rules in the original domain theory. During problem solving, each antecedent would only be pursued until it has reached the depth-bound in terms of rules in the original domain theory. This would prevent large amounts of time being consumed trying to use macro-rules and result in a fairer comparison of the performance of the learning and non-learning systems.

CHAPTER 8

GENESIS: SYSTEM OVERVIEW

This chapter concerns the organization of the complete GENESIS narrative processing system. GENESIS is built on top of the EGGS explanation-based learning system and enhances its ability to deal with plan-based explanations. The narrative understanding process in GENESIS is used to construct explanations for character's intentional behavior. EGGS then generalizes these explanations to produce plan schemata which GENESIS can then use to improve its understanding ability. Figure 8.1 illustrates the overall architecture of the system. The architecture of GENESIS is a straight-forward instantiation of the *prototypical explanation-based learning system* (PEBLS) as defined in [Segre87a] in which the *performance element* is the same as the *understander*. This chapter briefly summarizes the function of each of the components shown in Figure 8.1. The *schema library*, *understander*, *causal model*, *schema learner*, *word learner*, and *indexer/retriever* are discussed in detail in subsequent chapters.

8.1. The Parser and Lexicon

An adaptation of McDYPAR [Dyer83] (a micro-version [Schank81b] of the parser used in the BORIS system) is used to parse English sentences into predicate calculus wffs. The parser makes uses of word definitions stored in the *lexicon*. Numerous examples of the output of the GENESIS parser are given for the example narratives discussed in chapter 10 and appendix B. Since parsing is not the focus of the current research, no claims are made about the parser and alternative approaches could be used for this task (e.g. [Hirst87, Marcus80, Waltz84]).

8.2. The Understander, Schema Library, and Causal Model

The output of the parser is processed by the *understander* which attempts to construct explanations for characters' actions. The understander uses the system's existing knowledge of actions and inferences stored in the *schema library* in order to find connections between character's actions and infer the ultimate goals that motivated their actions. The embellished representation that it constructs for a narrative is referred to as the *causal model*. The *explanation* for a particular goal achievement is the subset of the causal model that supports the given goal state. The representation of actions used in GENESIS is an enhancement of the STRIPS-like representation presented in chapter 6 and is discussed in chapter 9. The techniques used in the understanding process are

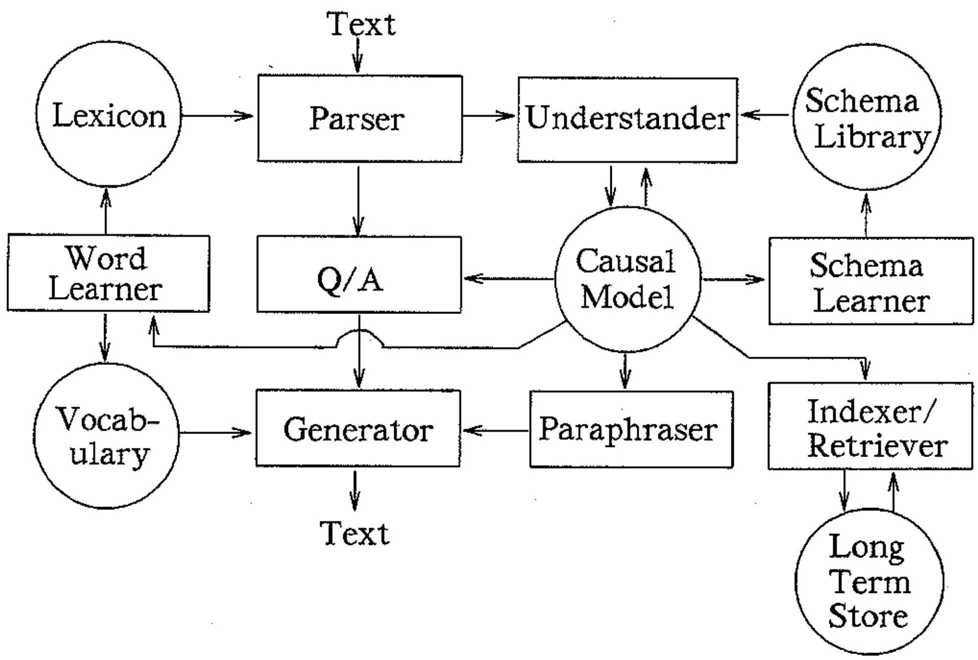


Figure 8.1: Architecture of GENESIS

similar to those used in previous work in narrative understanding [Charniak77, Cullingford78, DeJong82b, Dyer83, Schank81b, Wilensky83] and are discussed in chapter 10.

8.3. The Question-Answerer

A simple question-answerer is used to inspect the causal model built by the understander. Since the focus of the system is on the construction of generalizable explanations, this sub-system is primarily used to answer questions about *why* a character performed a certain action or *why* a particular state exists.

The question-answerer takes questions from the user after they have been parsed and employs a number of heuristics for retrieving an answer from the causal model. For example, one heuristic for answerering a question about why a character performed a particular action is to find another action that this action enabled and reply that the character executed the action so he could perform the subsequent action. The answer to the following question about the kidnapping narrative presented in chapter 2 was produced using this heuristic.

>Why did John approach Mary?

So John could aim the gun at Mary.

The *generator* is used to translate the question-answerer's replies into English.

Of course, there are many difficult issues involved in the retrieval of the most appropriate answers to questions that the question-answerer in GENESIS does not address (see [Lehnert78, Waltz78] for discussions of such issues). Consequently, its answers may sometimes seem a little strange or inappropriate.

8.4. The Paraphraser

A paraphrasing system is used to output information the system has obtained from its processing of a story. The purpose of this sub-system is not to concisely summarize the *plot* or the *point* of a narrative (as in [Lehnert82, Wilensky82]), but rather to simply convey the system's overall understanding of a piece of text.

If a narrative is recognized as an instance of a known schema or if a new schema is learned from the narrative, that schema is used to paraphrase the text. This is accomplished by printing out the instantiated actions composing the schema in chronological order. In addition, before printing out each action, any of its preconditions that have not previously been mentioned are also stated. The *generator* is used to compose English sentences for the assertions produced by the paraphraser. This technique for paraphrasing an instance of a schema is also used to produce English descriptions of general schemata (e.g. the English description of CaptureBargain shown in the trace in chapter 2). If a particular narrative cannot be interpreted as an instance of either an old or a new schema, the paraphraser simply generates English sentences for each of the individual actions in the text.

8.5. The Generator and Vocabulary

An adaptation of McMUMBLE [Schank81b] (a micro-version of the generator used in the TALE-SPIN narrative generating system [Meehan76]) is used to translate the output of the question-answerer and the paraphraser into natural language. Each predicate the system knows has a short program associated with it which instructs it how to state an instance of the predicate in English. These programs are stored in the *vocabulary*. Unfortunately, like many natural language systems, GENESIS' knowledge of words is separated into two different "lexicons," one for parsing and one for generating. A system that uses one lexicon for both tasks is presented in [Wilensky80].

8.6. The Schema Learner

The *schema learner* analyzes the causal model built by the understander in order to learn new plan schemata. It first monitors the causal model and detects when a character has achieved an important goal in a novel manner (i.e. in a manner for which the system does not already possess a schema). The explanations for how such goals are achieved can be used to learn new schemata. In addition to deciding when to learn, this sub-system includes an *explanation pruner*, *generalizer*, and *packager* as defined in chapter 3.[1] After the explanation for the goal achievement is extracted from the causal model, a special pruning function is used to remove actions and states that only support the goal through more abstract concepts. The pruned explanation is then generalized using the EGGS explanation generalization algorithm. Next, an enhancement of the procedure for building macro-operator discussed in section 6.2.2 is used to package the generalized explanation into a schema. Finally, this schema is stored in the schema library and indexed so that it can be used to aid the understanding of future narratives. The pruning, packaging, and indexing procedures as well as the procedure that decides when to learn are all discussed in detail in chapter 11.

8.7. The Word Learner

An additional feature that has been added to GENESIS is the ability to learn provisional definitions for unknown schema-related words. For example, as shown in the trace in chapter 2, GENESIS acquires, from a single narrative, definitions for "kidnapper," "ransom," and "kidnap" as well as a kidnapping schema. The word-learning process involves detecting the conceptual role that an unknown word fills in the schema that is being learned. After a word is associated with a slot in a new schema or with the new schema itself, a definition for the new word is constructed for both the lexicon and the vocabulary. This allows both the parser and generator to use the new word for, respectively, understanding narratives and answering questions. GENESIS' ability to learn new words is discussed in further detail in chapter 12.

8.8. The Indexer/Retriever and the Long Term Store

GENESIS also has the ability use the schemata it learns to index and retrieve specific narratives. After processing a narrative, the *indexer* stores the causal model constructed for this piece of text in the *long-term store* and indexes it under the most comprehensive schemata used in

[1]In PEBLS [Segre87a], the term *generalizer* is used to refer to the entire schema learning module; however, in GENESIS, this term is used solely for the EGGS explanation generalization process.

interpreting the story. When answering questions about a particular narrative, the *retriever* can be instructed to retrieve past episodes that are indexed under the same schema used to interpret the current text. These modules allow GENESIS to function as a conceptual retrieval system [Schank81a] which, during normal operation, automatically learns new ways to index events. Further details on this aspect of GENESIS are given in chapter 13.

8.9. The Importance of the Complete System

Although GENESIS is primarily a tool for exploring learning, this chapter has revealed that it contains a large number of components that have no direct connection to learning, such as a parser, question-answerer, paraphraser, etc.. An important question concerns the relevance of these additional components. While the non-learning components in GENESIS certainly make the learning system easier to test and debug, they also serve a more important purpose. The primary goal of learning is to improve the performance of an overall system in some relevant task domain. Several AI researchers have stressed the importance of viewing learning in the context of a performance system [Buchanan77, Mitchell83, Simon83]. The requirement of improving overall system performance often imposes important constraints on a learning system. As a result, certain degrees of freedom are eliminated that could otherwise be unfairly manipulated to bias the performance of an isolated learning system.[2]

For example, in GENESIS, the same explanations that support learning must also be used to answer questions about the text, and the schemata produced by the learning mechanism must be able to aid the understanding of future narratives as well as support reasonable paraphrases. Many representations suitable for learning are unsuitable for performing these other tasks. Most other EBL systems, on the other hand, do not use the explanations they construct for any task other than to support generalization. In fact, most do not even retain the entire generalized explanation, but rather extract only a macro-rule or macro-operator that summarizes the overall preconditions and post-conditions. These data structures do not generally contain information on the sub-structure that composes and justifies them.[3] In GENESIS, the entire generalized explanation is retained in order to be able to use the learned schema to understand future narratives and answer questions about them.

---

[2]A similar point is made in [Falkenhainer87 (section 4.1)].

[3]The generalized triangle tables retained by STRIPS do contain much of this information. This knowledge is used to detect and correct problems encountered during execution monitoring.

Complementing the impact that the understanding system has on the learning system is the impact that the learning system has on the understanding system. Building explanations sufficient to support generalization and schema acquisition is an important test of how well a text processing system has actually "understood" a piece of text. Previous tasks that have been used to test understanding are question-answering [Lehnert78], paraphrasing [Cullingford78, DeJong82b], and summarizing [Lehnert82]. Unlike these tasks, the ability to construct an explanation sufficient for generalization is a crucial test of how well the system has comprehended the global causal structure of a piece of text. Answering *why* and *how* questions are good ways of testing comprehension of *local* causal structure; however, unlike question-answering, generalization requires the construction of a globally consistent causal model. Paraphrasing and summarizing require having an adequate global structure for the text; however, they do not generally test the understanding of *causal* structure. For example, FRUMP [DeJong82b] produced summaries of newspaper articles using *sketchy scripts* which contain only temporally ordered lists of events with very little causal information. In summary, for many narratives (e.g. the examples in this thesis), it is fair to say that a system has not truly "understood" the text unless it has the ability to produce an explanation that could be generalized into a new schema using EBL techniques. Therefore, the ability to learn a schema from a narrative can be viewed as an additional test of the understanding abilities of a natural language system.

CHAPTER 9

GENESIS: SCHEMA REPRESENTATION

All domain specific knowledge in GENESIS is represented in a declarative fashion in the schema library. This knowledge is divided into information about *objects*, *attributes*, *states*, and *actions*. This chapter is comprised of separate discussions of the representations used for each of these classes of knowledge. A complete listing of all of GENESIS' initial knowledge is given in appendix C.

9.1. Objects

An *isa hierarchy* is used to specify a taxonomy of object classes. The complete GENESIS object hierarchy is shown in Figure 9.1. [1] The system uses standard path-finding inheritance techniques to determine class membership [Charniak85, Nilsson80]. If a path is found, the system constructs a proof of class membership based on the path. This proof is composed of inference rules such as:

Isa(?x, Gun) → Isa(?x, Weapon)

For efficiency reasons, this specialized inference technique is preferred to giving the EGGS theorem prover explicit axioms specifying the taxonomy and allowing it to construct proofs of class membership. Since queries regarding class membership are quite frequent, a specialized strategy for these queries substantially increases the overall efficiency of the system.

9.2. Attributes and States

Attributes are facts about objects that are *intrinsic* [Kedar-Cabelli87b] and are not changed by actions, while *states* are facts about objects that are affected by actions. GENESIS' hierarchies of attributes and states are shown in Figure 9.2 and Figure 9.3, respectively. Like the object isa-hierarchy, these hierarchies support abstraction inferences such as:

Father(?x,?y) → Parent(?x,?y)

Knowledge of attributes and states also includes backward-chaining rules for inferring instances of a predicate and forward-chaining rules for making inferences from an instance of a predicate. For

[1] Special notice should be taken of the distinction between a Character and a Person. A Person is a human being while a Character is any agent that is capable of executing a volitional action and includes Persons as well as collective agents such as Companies.

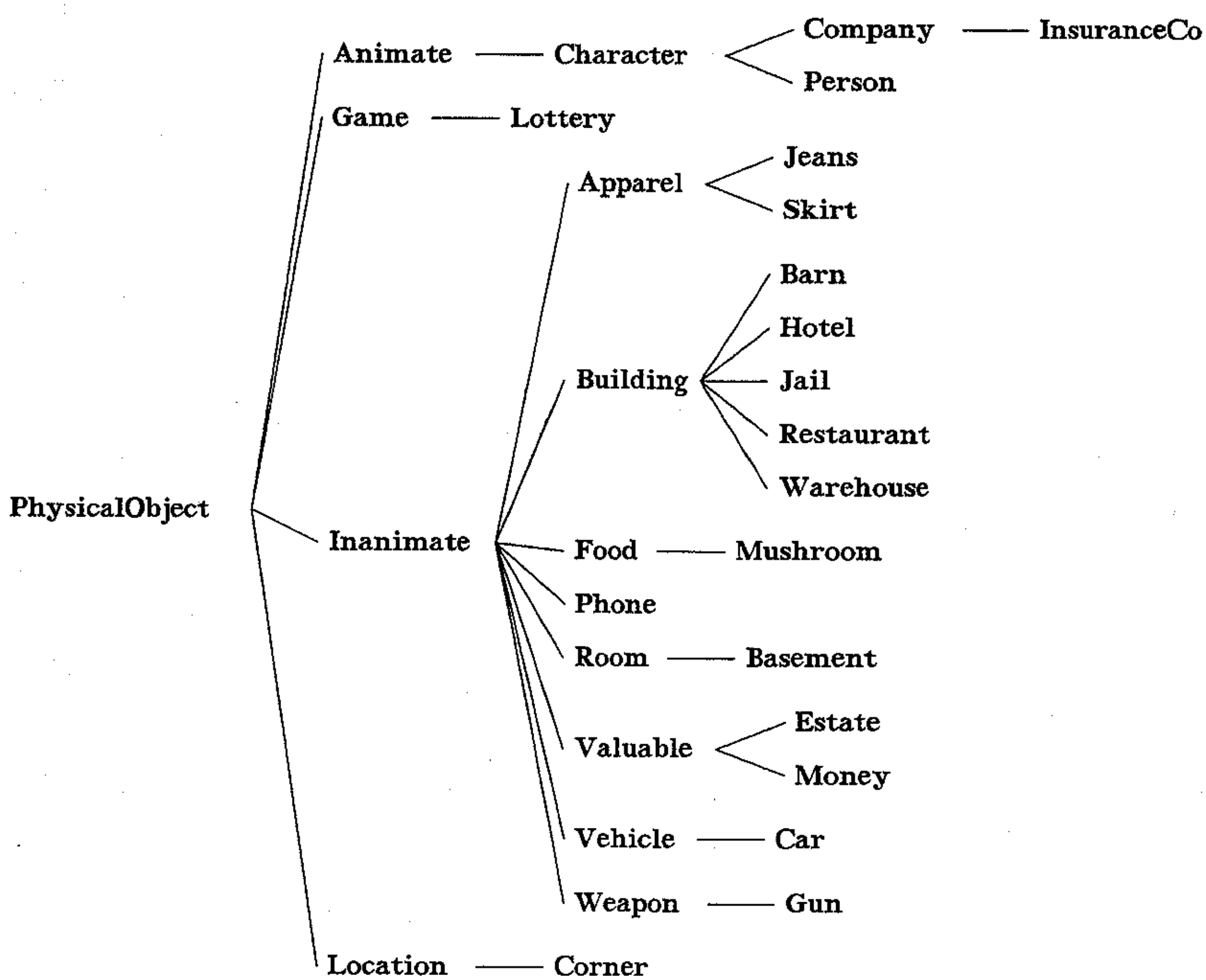


Figure 9.1: GENESIS Hierarchy of Objects

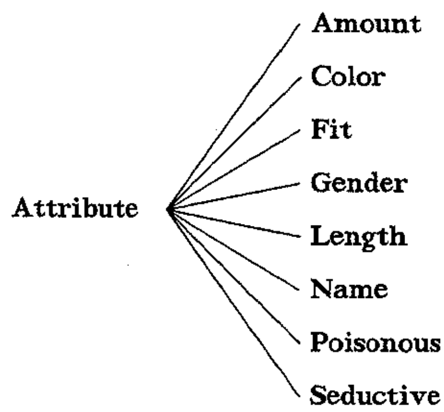


Figure 9.2: GENESIS Hierarchy of Attributes

example, the following rule is used in a backward-chaining fashion to infer that someone possesses a particular amount of an object if he possesses a larger amount of that object:

Amount(?y,?g,?u) ∧ Possess(?a,?y) ∧ Amount(?x,?l,?u) ∧ Isa(?x,?t) ∧ Isa(?y,?t) ∧
LessThan(?l,?g) → Possess(?a,?x)

In this rule, Amount(?x,?l,?u) specifies that the object ?x consists of a number, ?l, of units, ?u (e.g. Amount(Money1, 100, dollars)). All of the rules attached to states and attributes in GENESIS are given in section 2 of appendix B.

9.3. Actions

Actions specify dynamic changes in the state of the world and are also arranged in an abstraction hierarchy. A graph of the complete GENESIS action hierarchy is shown in Figure 9.4. Actions are divided into *volitional actions*, such as Murder, which are goal directed actions willingly executed by an actor, and *non-volitional actions*, such as Die.

Actions are represented using an enhancement of the STRIPS-like representation discussed in section 6.1. The first modification is that *preconditions* are divided into *constraints*, *preconditions*, and *motivations*. *Constraints* are required attributes or classes of the arguments of an action and therefore can not be achieved by other actions. *Preconditions* are states that enable an action and can be achieved by other actions. *Motivations* are mental states of the actor, such as goals, goal

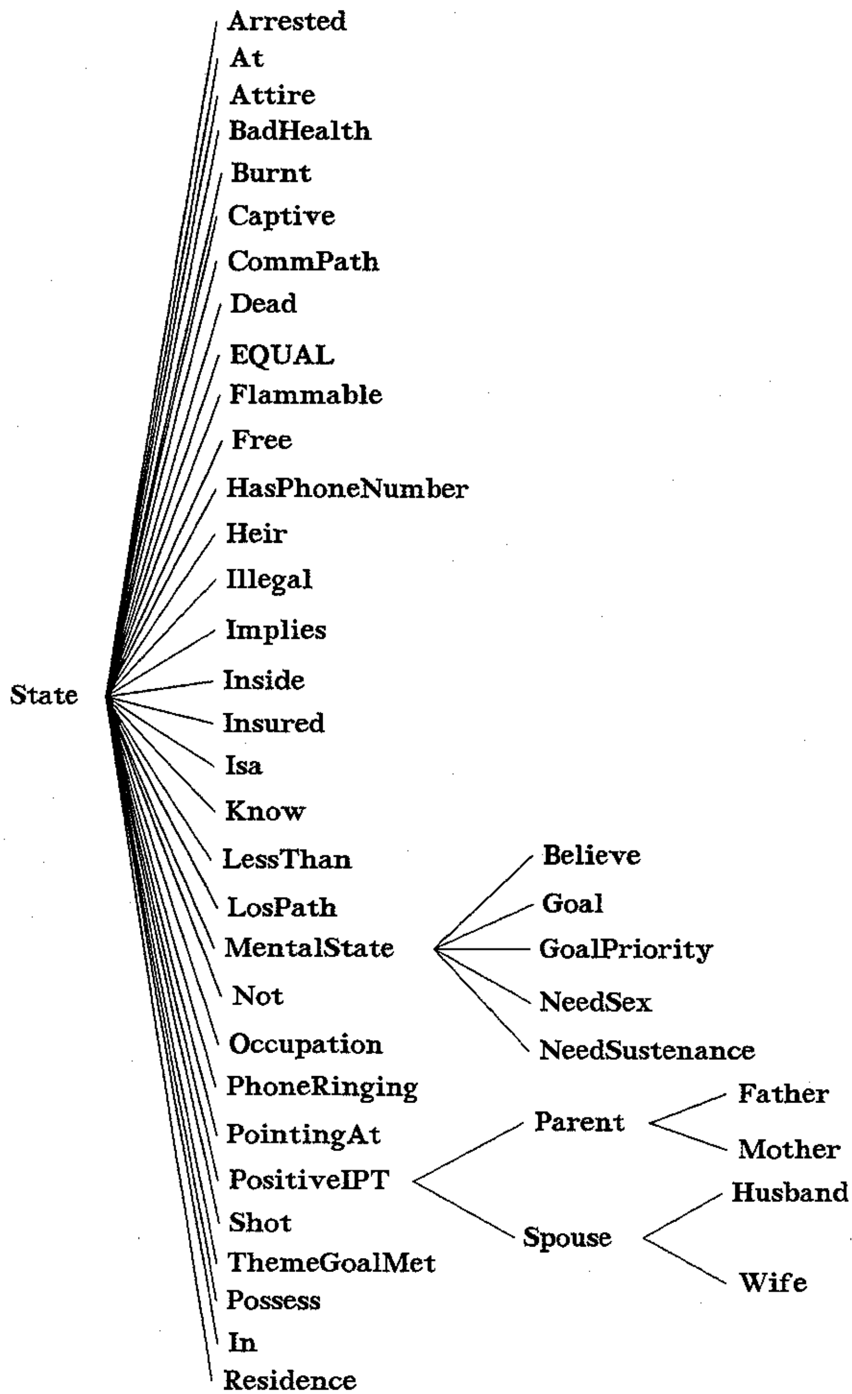


Figure 9.3: GENESIS Hierarchy of States

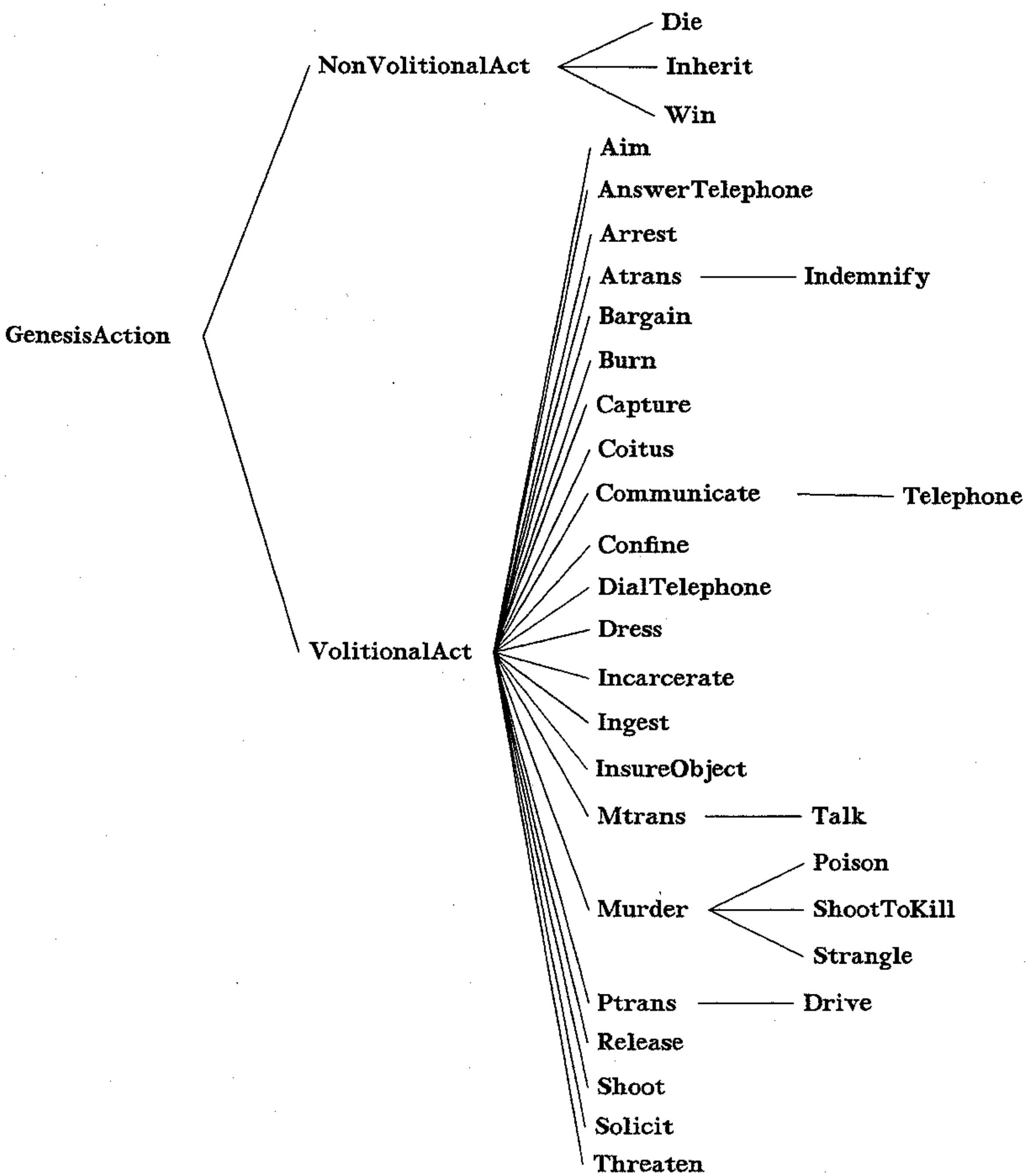


Figure 9.4: GENESIS Hierarchy of Actions

priorities, and beliefs, that motivate him to perform a volitional action.[2] Together, constraints, preconditions, and motivations are called the *supports* of an action. As in the representation in chapter 6, *effects* are states, possibly negated ones, resulting from the execution of an action. An action definition includes only the supports and effects unique to that particular action. Supports and effects that are inherited from more abstract actions are defined at that level. As an example, the supports and effects of the Murder action and one of its specializations (Poison) are shown in Table 9.1. The effect stating that the victim is dead is specified only in the definition of Murder. The preconditions that the actor must possess a poisonous food and the victim must be hungry and the effects that the poisonous food is inside the victim and no longer possessed by either the actor or victim are specific to Poison and are specified only in its definition.

Knowledge about an action may also include information about its *expansion*, i.e. its decomposition into more primitive actions. An action can be recursively defined as a macro-operator that has an expansion in terms of other actions (which may also be macro-operators). The expansion of an action is also analogous to the body of a *script* [Schank77] or MOP [Schank82].[3] The complete set of action specifications in GENESIS should not be interpreted as defining an ultimate set of primitive actions. Action definitions that do not include expansions are not necessarily primitive actions; rather, they should generally be interpreted as currently lacking a complete definition. Although such action definitions must eventually bottom out in a set of primitives, sets that have been proposed have many inadequacies [Wilks75]. Therefore, in GENESIS, no commitment is made regarding an ultimate set of primitives.

Table 9.1: Sample Definition of Action Supports and Effects

Action	Constraints	Preconditions	Effects
Murder(?a,?v) "?a murdered ?v"	Isa(?a,Character) Isa(?v,Person)		Dead(?v)
Poison(?a,?v,?p) "?a poisoned ?v with ?p"	Poisonous(?p) Food(?p)	Possess(?a,?p) NeedSustenance(?v)	Inside(?p,?v) ¬Possess(?a,?p) ¬Possess(?v,?p)

[2]The distinction between preconditions and motivations is used by the question-answerer when determining the appropriate answer to a *why* question.

[3]Scripts are defined directly in terms of primitive conceptual dependency actions [Schank75] while MOPs, like our actions, are defined recursively.

The expansion of an action schema includes *subactions*, *internal states*, *links*, *abstractions*, and *ordering constraints*. *Subactions* are the set of more primitive actions which compose the action. *Internal states* (also referred to as *internals*) are states that are true sometime during the execution of an action but are neither preconditions nor effects. *Links* define the relationships among the subactions, supporters, and internal states, and thereby specify the causal structure of the expansion. *Abstractions* specify the relationships between facts in the expansion and facts in the expansions of more abstract actions. *Ordering constraints* (also referred to as *orderings*) are additional constraints on the temporal ordering of the subactions that are not implicit in the support structure specified by the links. These are needed to avoid protection violations as discussed in section 6.2.2. As an illustration of expansion definitions, the expansions of Murder and Poison are shown in Table 9.2 and Table 9.3, respectively. A link of the form: (Effect (Subaction 2)(Effect 1)) specifies that the first effect of the action is an effect of the action's second subaction. Other links are interpreted in a corresponding fashion. An abstraction of the form ((Subaction 3) Murder (Subaction 2)) specifies that the third subaction of the action corresponds to or is a specialization of Murder's second subaction. Other abstractions are interpreted in a corresponding manner.

Table 9.2: Expansion of Murder

Subactions	Internals	Links
GenesisAction(?a) Die(?v)	State(?v) State(?o) BadHealth(?v)	(Effect (Subaction 1)(Internal 1)) (Antecedent (Internal 3) (Internal 1)) (Antecedent (Internal 3) (Internal 2)) (Precondition (Subaction 2) (Internal 3)) (Effect (Subaction 2)(Effect 1))

Table 9.3: Expansion of Poison

Subactions	Internals	Abstractions
Atrans(?a,?p,?v,?l) Ingest(?v,?p) Die(?v)	Possess(?v,?p) BadHealth(?v)	((Subaction 1) Murder (Subaction 1)) ((Subaction 3) Murder (Subaction 2)) ((Internal 1) Murder (Internal 1)) ((Effect 1) Murder (Internal 2)) ((Internal 2) Murder (Internal 3))
Links		
(Precondition (Subaction 1) (Precondition 1)) (Effect (Subaction 1) (Effect 2)) (Precondition (Subaction 2) (Internal 1)) (Effect (Subaction 2) (Effect 1)) (Antecedent (Internal 2) (Constraint 1)) (Precondition (Subaction 3) (Internal 2)))	(Effect (Subaction 1) (Internal 1)) (Constraint (Subaction 2) (Constraint 2)) (Motivation (Subaction 2) (Precondition 2)) (Effect (Subaction 2) (Effect 3)) (Antecedent (Internal 2) (Effect 1))	

A graphical display of the expansions of Poison and Murder and how they relate to each other is shown in Figure 9.5. The expansion of Murder can be summarized as follows: An actor, ?a, performs some action that causes a victim, ?v, to be in a state that implies he is in a state of bad health which in turn enables the victim's non-volitional death. The expansion of Poison can be summarized thus: An actor, ?a, gives a poisonous food, ?p, to a victim, ?v, which results in the victim possessing the food. Since the victim is hungry and has the food, he eats the food that results in the food being inside him. The fact that the food is poisonous and inside him implies that the victim is in a state of bad health. This in turn enables his non-volitional death. The actor's action of giving the victim poisonous food corresponds to the actor's unspecified action in the expansion of Murder while the state of bad health and death correspond to these same parts in Murder's expansion. The relationships between facts and more general abstractions are represented as logical implications. For example:

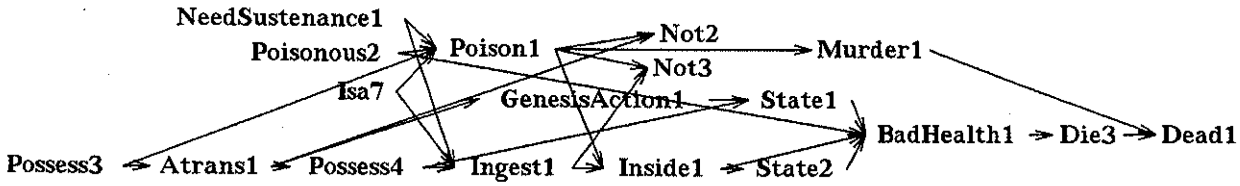
Poison(?a, ?v, ?p) → Murder(?a, ?v)

Atrans(?a, ?p, ?v, ?l) → GenesisAction(?a)

It should be noted that motivations of other characters' actions in the expansion of a volitional schema are always preconditions of the overall schema or the effects of actions performed by the volitional actor of the schema. For example, since the fact that the victim is hungry motivates him to ingest the poion, the victim being hungry is a precondition of the volitional Poison schema. The simplifying assumption is made that other actors will always perform their subactions if the motivations for these actions are met. This insures that all of the effects of the action will be achieved if the volitional actor performs all of his actions.

A final type of information attached to actions is *suggestions*. A part of each action definition is a list of larger schemata of which that action is a subaction. An action is said to *suggest* schemata of which it may be a part and which therefore may provide a reason for its execution. This process forms a crucial part of the understanding mechanism and is discussed in detail in the following chapter.

In conclusion, it should be noted that the representation presented in this section has been used to represent a variety of actions needed for the understanding of narrative text. Numerous additional examples of action definitions using this representation are presented in section 1 of appendix C.



Dead1	Dead(?v)	?v is dead.
Murder1	Murder(?a,?v)	?a murdered ?v.
Poison1	Poison(?a,?v,?p)	?a poisoned ?v with ?p
Isa7	Isa(?p,Food)	?p is a food.
Poisonous2	Poisonous(?p)	?p is poisonous.
NeedSustenance1	NeedSustenance(?v)	?v is hungry.
Possess3	Possess(?a,?p)	?a had ?p.
Inside1	Inside(?p,?v)	?p is inside ?v.
Ingest1	Ingest(?v,?p)	?v ate ?p.
Possess4	Possess(?v,?p)	?v had ?p.
Atrans1	Atrans(?a,?p,?v,?12)	?a gave ?v ?p.
Not2	¬Possess(?a,?p)	?a does not have ?p.
Not3	¬Possess(?v,?p)	?v does not have ?p.
Die3	Die(?v)	?v died.
BadHealth1	BadHealth(?v)	?v is in bad health.
State2	State(?p)	?p is in some state.
State1	State(?v)	?v is in some state.
GenesisAction1	GenesisAction(?a)	?a did something.

Figure 9.5: Expansions of Murder and Poison

CHAPTER 10

GENESIS: NARRATIVE UNDERSTANDING

A substantial body of research in natural language processing has addressed the problem of "understanding" narrative text. Most of the research in narrative understanding has focussed on constructing explanations for character's actions and inferring missing actions that are only implicitly mentioned [Charniak85 (chapter 10), Schank81b]. Although these are difficult problems which are far from completely solved, a reasonable amount of progress has been made and a number of useful mechanisms have been developed.

In the work on narrative understanding, a distinction can be made between *schema-based* (also called *script-based*) and *plan-based* understanding mechanisms [Schank77]. A *schema-based* understanding mechanism attempts to directly and efficiently access a relatively specific knowledge structure (a schema) that accounts for the actions in the text. This schema can then be efficiently used to causally connect actions in the narrative and fill in missing actions. In terms of problem solving, it is analogous to using a macro-operator to solve a particular problem in one step. Examples of systems that use schema-based understanding are SAM [Cullingford78] and FRUMP [DeJong79].

A *plan-based* understanding mechanism, on the other hand, can be used for understanding novel situations. It involves searching for a set of missing actions that causally connect to observed actions to form a plan that achieves a character's goal. In terms of problem solving, this approach is more like doing search-intensive planning to achieve a goal. In Newell's terminology [Newell73], plan-based understanding is a *weak method* while schema-based understanding is a *strong method*.

PAM [Wilensky78] is an example of a system that does plan-based understanding. Although planning and plan-based understanding can make use of the same database of actions, they are quite different processes. In planning, one is given an initial state and a goal and must search for a plan that can be executed in the initial state and that achieves the goal. In plan-based understanding, one is given a set of observed actions and must search for a plan that includes or "covers" these actions and that achieves a character's goal. In understanding, goals must usually be inferred since they are rarely explicitly mentioned.

Since a robust understanding system must be able to deal with both mundane and novel situations, several recent understanding systems perform both schema and plan-based understanding. For example, BORIS [Dyer83] and FAUSTUS [Norvig83, Wilensky83] are systems that use both of these understanding mechanisms. GENESIS also employs both schema and plan-based mechanisms in its understanding process. First, the system tries to find an action schema that will directly explain the characters' actions. If this fails, it resorts to trying to causally connect individual actions in a plan-based manner. However, the search it performs during plan-based understanding is very limited in order to prevent it from spending an inordinate amount of time performing combinatorially explosive search through the space of all possible explanations. Consequently, GENESIS cannot produce adequate explanations for certain narratives although it theoretically could understand them given an exhaustive search algorithm and unlimited time and space. Nevertheless, the system's limited ability to do plan-based understanding allows it to construct explanations for many novel plans presented in narratives. The system then uses the EGGS generalizer to perform explanation-based learning on these explanations in order to learn new schemata which allow it to understand future instances of the plan using schema-based understanding. Explanation-based learning in GENESIS can therefore be viewed as the acquisition of schemata that allow the system to process narratives using efficient schema-based techniques which previously could only have been understood using inefficient, search-intensive, plan-based understanding.

In order to understand exactly how learning can improve performance and allow GENESIS to construct explanations for narratives that it previously could not comprehend, detailed knowledge of the understanding process is needed. The remainder of this chapter examines the schema and plan-based understanding mechanisms in GENESIS. A narrative used in the acquisition of a "murder for inheritance" schema (an example due to Paul O'Rorke and introduced in [DeJong83]) is used to illustrate these processes. The English text and parsed version for this narrative are shown in Figure 10.1[1] and additional action definitions needed for this narrative are shown in Table 10.1. A detailed system trace for this example is presented in appendix D. The reader may find it helpful to refer to this trace while reading subsequent passages describing the processing of this example.

10.1. Schema Selection

If a schema-based mechanism is to be able to process a broad range of texts, it must have access to a large number of schemata. Therefore, in order to avoid repeated searching through the entire database of schemata, it must also have an efficient method for selecting the particular

[1] Variables in parser output should be interpreted as being existentially quantified.

Claudius was Agrippina's husband and owned an estate.
Agrippina gave him a poisonous mushroom and he died.
She inherited the estate.

Isa(Person1,Person), Gender(Person1,male), Name(Person1,Claudius), Isa(Person2,Person), Gender(Person2,female), Name(Person2,Agrippina), Isa(Estate1,Estate), Husband(Person1, Person2), Possess(Person1,Estate1), Poisonous(Mushroom1), Isa(Mushroom1,Mushroom), Atrans(Person2,Mushroom1,Person1,?AT6), Die(Person1), Inherit(Person2,Estate1,?FROM3)

Figure 10.1: English and Parsed Versions of the Learning Narrative for the Murder Example

Table 10.1: Additional Action Definitions for the Murder Example

Action	Constraints	Preconditions	Effects	Suggestions
Atrans(?a,?o,?t,?l) "?a gave ?t ?o at ?l"	Isa(?a,Character) Isa(?o,Inanimate) Isa(?t,Character) Isa(?l,Location)	Possess(?a,?o)	Possess(?t,?o) ¬Possess(?a,?o)	Atrans(?a,?o,?t) ⇝ Poison(?a,?t,?o)
Die(?a) "?a died"	Isa(?a,Person)		Dead(?a)	
Inherit(?a,?o,?d) "?a inherited ?d's ?o"	Isa(?a,Character) Isa(?d,Person) Isa(?o,Inanimate)	Heir(?a,?d) Possess(?d,?o) Dead(?d)	Possess(?a,?o) ¬Possess(?d,?o)	

schemata that are applicable to the current input. This process is frequently referred to as *schema selection* or *frame selection*. It is a difficult problem and has been the subject of several research efforts [Charniak78, Charniak82, DeJong82b, Lytinen84, Norvig83].

In GENESIS, the process of selecting a schema has two stages: *suggestion* and *determination*. An action A in a narrative *suggests* a schema B if A is a subaction of B. Each suggested schema is *instantiated* (i.e. a unique instance is created with the appropriate variable bindings) and monitors the inputs in order to find confirming evidence. When a schema has found enough confirming evidence, it is *determined* and its complete expansion is added to the causal model.[2] An early discussion of this approach and some of its advantages was presented in [DeJong84]. The same basic technique was also used in the original implementation of GENESIS [Mooney85a, Mooney85b].

[2]The term *determination* is taken from [Norvig83] where it is used to refer to a similar process in FAUSTUS. *Suggestion* in GENESIS corresponds to the combination of *invocation* and *instantiation* as defined in [Norvig83].

10.1.1. Schema Suggestion

A specification of the suggestion process is shown in Figure 10.2. As mentioned in the previous chapter, an action definition includes suggestions of relevant schemata. For example, the definition of Atrans given in Table 10.1 includes the following suggestion:

Atrans(?a,?o,?t,?l) $\rightsquigarrow$ Poison(?a,?t,?o)

where a statement of the form: A $\rightsquigarrow$ B is read: A suggests B. Therefore, the input: Atrans(Person2, Mushroom1, Person1, ?AT6) in the murder story potentially suggests the schema: Poison(Person2, Person1, Mushroom1). A suggestion is not complete, however, until it is confirmed that the suggested instance obeys all of the constraints on the schema. For the example, the system must first prove the following facts using the deductive retriever in EGGS.

Poisonous(Mushroom1), Isa(Mushroom1, Food), Isa(Person1, Character), Isa(Person2, Person)

The first and fourth constraints were given explicitly in the input and the second and third ones are easily inferred using the object hierarchy. Therefore, the instantiated Poison schema is added to the list of suggested schemata where it is monitored for subsequent determination.[3] The final step in

---

```
procedure SuggestSchemata(i)
(* Suggest schemata for input i *)
  for each suggestion a ~> b in the action definition of i do
    if i unifies with a with MGU θ
      then
        let s = bθ
        let q = true
        for each constraint c of s do
          if c does not have any variables ∧ ¬Retrieve(c)
            then let q = false
        if q
          then
            put s on the list of suggested schemata
            for each super b of action s in the action hierarchy do
              put b on the list of suggested schemata
```

Figure 10.2: Procedure for Suggesting Schemata

---

[3]For this particular example, it may seem that the input Poisonous(Mushroom1) should directly suggest Poison instead of being checked by a suggestion initiated by the Atrans action. However, in order to keep the suggestion mechanism simple, only subactions suggest higher-level schemata that can explain them. Allowing constraints to suggest schemata would greatly complicate the process of learning suggestions. Having all constraints suggest a schema would result in inputs such as

the suggestion of a schema is the suggestion of its generalizations in the abstraction hierarchy. For example, when the Posion schema is suggested in the example narrative, it also causes Murder(Person2, Person1) to be suggested.

In general, a suggested instance of a schema may contain existentially quantified variables. For example, if the arguments to Poison had included a location, the variable used to denote the location of the Atrans (?AT6) would be included in the suggested schema. Constraints on such variables (e.g. Isa(?AT6, Location)) are not checked during suggestion; however, any constant that is later matched to such variables must satisfy these constraints. If the EGGS' deductive retriever were used during suggestion to retrieve constraints with variables, it would lead to premature assignment of unknown schema variables. In the example, ?AT6 could be immediately associated with any previously mentioned location..

10.1.2. Schema Determination

A suggested schema continually checks inputs in the narrative to see if they match facts in its expansion. This process of examining suggested schemata for determination is described in Figure 10.3. Each fact in the expansion of a suggested schema is checked to see if it matches the input fact. If an input matches a suggested fact and all new schema variable bindings resulting from this match satisfy the schema constraints, then these variable bindings are applied to the schema expansion.[4] The first input that is checked by this process is the input that suggested the schema. In the example, this is the Atrans input and it matches a subaction in the expansion of the Posion schema (shown in Figure 10.4).

Each time a fact in the expansion of a suggested schema is matched to an input, it is marked as having been "found." In addition, facts related to the "found" fact are assumed to be true and also marked as "found." When an action is marked as "found", all of its effects, preconditions, motivations, and constraints are also marked. This is because if an action has occurred, then all of its supports must have been true and all of its effects must now be true. When a fact is marked as "found", all of its antecedents (if it has any) are also marked. If it is the antecedent of another fact, all of whose antecedents are now marked, this fact is also marked since it must be true if all of its

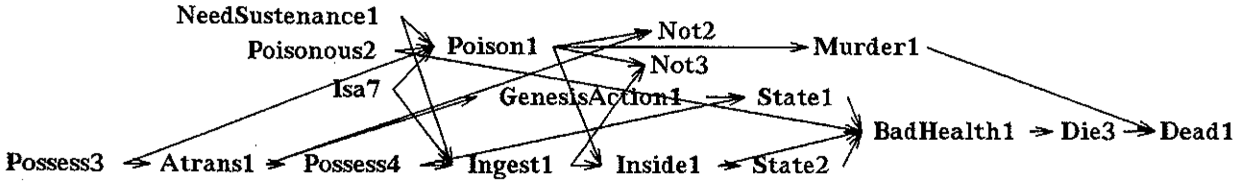
Isa(Person2, Person) and Isa(Mushroom1, Food) suggesting Poison, which is clearly worse. Distinguishing between constraints such as Poisonous(Mushroom1) and Isa(Mushroom1, Food) would require a complex analysis of the frequency of occurrence of such constraints.

[4]In FAUSTUS, the corresponding process of determining the values of additional frame slots after instantiation is called *elaboration* [Norvig83].

```
procedure CheckSuggestedSchemata(i)
(* Check if input i is a part of a suggested schema *)
  for each suggested schema s do
    for each fact f in the expansion of s do
      if f and i unify with MGU θ ∧
        any newly bound schema variables in θ satisfy the schema constraints
        then
          for each fact g in the expansion of s do
            replace g with gθ
          AssumeFound(f)
          if all the subactions in the expansion of s are "found"
            then determine s (i.e. add its expansion to the causal model)

procedure AssumeFound(f)
  if f is not already marked as "found" then
    mark f as "found"
    if f is an action
      then
        for each support b and each effect b of f do AssumeFound(b)
      else
        for each antecedent a of f do AssumeFound(a)
        for each consequent c of which f is an antecedent do
          let q = true
          for each antecedent a of c do
            if a is not "found" then let q = false
          if q then AssumeFound(c)
        for each action a of which f is an effect do
          let q = true
          for each effect e of a do
            if e is not "found" then let q = false
          if q then AssumeFound(a)
    if f has an abstraction b or is an abstraction of b
       where b is in the expansion of another suggested schema
      then AssumeFound(a)
```

Figure 10.3: Procedure for Checking a Schema for Determination



Dead1	Dead(Person1)	Claudius is dead.
Murder1	Murder(Person2,Person1)	Agrippina murdered Claudius.
Poison1	Poison(Person2,Person1,Mushroom1)	Agrippina poisoned Claudius with the mushroom.
Isa7	Isa(Mushroom1,Food)	The mushroom is a food.
Poisonous2	Poisonous(Mushroom1)	The mushroom is poisonous.
NeedSustenance1	NeedSustenance(Person1)	Claudius is hungry.
Possess3	Possess(Person2,Mushroom1)	Agrippina had the mushroom.
Inside1	Inside(Mushroom1,Person1)	The mushroom is inside Claudius.
Ingest1	Ingest(Person1,Mushroom1)	Claudius ate the mushroom.
Possess4	Possess(Person1,Mushroom1)	Claudius had the mushroom.
Atrans1	Atrans(Person2,Mushroom1,Person1,?12)	Agrippina gave Claudius the mushroom.
Not2	¬Possess(Person2,Mushroom1)	Agrippina does not have the mushroom.
Not3	¬Possess(Person1,Mushroom1)	Claudius does not have the mushroom.
Die3	Die(Person1)	Claudius died.
BadHealth1	BadHealth(Person1)	Claudius is in bad health.
State2	State(Mushroom1)	The mushroom is in some state.
State1	State(Person1)	Claudius is in some state.
GenesisAction1	GenesisAction(Person2)	Agrippina did something.

Figure 10.4: Expansions of Suggested Poison and Murder Schemata

antecedents are. If a marked state is the effect of an action, all of whose positive effects are now marked, this action is also marked. This results in assuming suggested actions whose effects are all known (or assumed) to be true. Finally, any facts in the expansions of other suggested schemata that are abstractions or specializations of a marked fact are also marked as "found." Whenever all the subactions in the expansion of a suggested schema are all marked as "found," that schema is *determined*. This adds the suggested action as well as its supports, effects, and expansion to the causal model where they become part of the system's "interpretation" of the text.

In the murder example, when Atrans1 (see Figure 10.4) is found, it causes precondition Possess3, precondition Not2, effect Possess4, and abstractions State1 and GenesisAction1 all to be marked as "found". The next input in the example is: Die(Person1). This input matches the fact Die3 in the expansions of both the Poison and Murder schemata. Although it does not result in any new variable bindings, it causes all of the other facts in the expansions of Murder1 and Poison1 to be marked. In particular, it marks the fact that Claudius ate the mushroom (Ingest1) because he must have been sick (BadHealth1) as a precondition of dying (Die3) and this sickness can be inferred from the fact that the mushroom is poisonous (Poisonous2) and inside him (Inside1) which in turn is the only positive effect of the Ingest. Once this missing action is inferred, the Poison schema is determined since all of its subactions (Atrans1, Ingest1, and Die3) have either been found in the inputs or assumed to have occurred. The determination process simply adds all of the facts shown in Figure 10.4 to the causal model.

10.1.3. Comparison to Other Approaches to Schema Selection

Although GENESIS' understanding system is primarily intended as an implementation of established techniques rather than original research in narrative understanding, its approach to schema selection is somewhat different from previous methods. In general, its approach is more conservative and gradual than previous approaches.

Both SAM [Cullingford78] and FRUMP [DeJong79] selected a schema based on the conceptual representation of a single input. SAM worked by "...introducing the 'largest,' most inclusive script it possesses which is initiated by the first conceptualization" [Schank81b (p. 109)]. Although FRUMP had many more scripts than SAM and was not tied to activating one from the first input, it also used a single conceptualization to determine a relevant script. A discrimination net was used to index relevant scripts based on the content of a single conceptual dependency form [DeJong79]. The problem with this approach is that it may prematurely decide on a schema that does not actually account for the rest of the inputs. Since neither SAM nor FRUMP was able to backtrack and consider alternative interpretations, such a mistake results in misunderstanding the text. More recent work has addressed the issue of backtracking and considering alternative interpretations after an initial schema fails to account for later inputs [Granger80, O'Rorke83, Orejel-Opisso84]. In addition, psychological studies indicate that humans perform backtracking of this sort when understanding certain stories [Collins80].

Separating schema selection into an initial suggestion process followed by a confirming process of determination is an attempt to minimize misunderstanding and the need for backtracking by being more cautious when selecting an initial schema. The system does not commit itself to a particular schema (i.e. it is not determined) until it is clear that it can account for other inputs in the narrative. Also, the process of inferring missing actions that complete a *causal chain* [Schank81b] is integrated with the process of checking a schema for determination. A schema is determined when all of its subactions have been observed or can be inferred as filling gaps in a causal chain of events. However, once a schema is determined, GENESIS is incapable of retracting it and cannot backtrack to consider alternative interpretations.

Several other recent understanding systems have also taken a more cautious approach to initial schema selection. As in GENESIS, the basic approach is to insure that a schema accounts for a number of inputs before actually selecting it. FAUSTUS has three stages in its process of selecting a frame. A frame is first *invoked* then *instantiated* and finally *determined* [Norvig83]. WIMP [Charniak86] uses spreading activation to "suggest" relevant frames. A process based on Occam's Razor then selects the suggested frame that accounts for the most inputs while making the least number of assumptions. A plan recognition technique based on non-monotonic deduction is discussed in [Kautz86]. It also uses an "Occam's Razor" approach which picks the least number of plans needed to account for all of the inputs. The understanding system used to explain observed assembly sequences in the ARMS robotics system does not determine a schema until *all* of its subactions have been directly observed [Segre87a]. This is possible in the ARMS domain since one is guaranteed to observe a complete sequence of low-level operators. In narrative understanding, one must have some way of inferring missing actions since stories rarely (if ever) mention all of the actions that took place.

10.2. Causally Connecting Actions Without a Schema

If, while processing a narrative, GENESIS encounters an input action that does not match part of an existing or suggested schema nor suggest any schemata, it attempts to explain the action in a more plan-based fashion by attempting to connect the new action to the effects of previous actions. The procedure GENESIS uses to connect actions without a schema is an elaboration of the plan verification algorithm given in section 6.2.1. Since, in GENESIS, this process is only attempted when schema-based understanding cannot be used to explain an action, a description of it is embedded in the complete understanding process shown in Figure 10.5.

The implicit goal of the understanding procedure is to determine an action or connected set of actions taking place in the narrative that achieve a character's ultimate goal. Since narratives do not always explicitly reveal characters' ultimate goals, the system must have a way of inferring them. An aspect of Schank and Abelson's theory of *goals* and *themes* [Schank77] is used to determine such goals. In their view, a goal arises either as a subgoal of a plan to achieve a higher level goal, or as a result of a theme. Schank and Abelson define themes as: "containing the background

```
for each parsed input i do
   if i is not an action
      then ProcessFact(i)
      else ProcessAction(i)

procedure ProcessFact(i)
   if ¬Retrieve(i)
      then
         Assert(i)
         CheckSuggestedSchemata(i)
         if i does not match part of a suggested schema
            then
               let g = Retrieve(ThemeGoalMet(?x, i))
               if g then call schema learner on explanation for g

procedure ProcessAction(i)
   Assert(i)
   CheckSuggestedSchemata(i)
   if i does not match part of a suggested schema
      then
         if i does not achieve a thematic goal
            then
               SuggestSchemata(i)
               CheckSuggestedSchemata(i)
         if i does not suggest any schemata
            then
               for each support s of i do
                  if s is a constraint with variables
                     then let r = false
                     else let r = Retrieve(s)
                  let θ be the MGU of s and r
                  if r ∧ any newly bound schema variables in θ
                        satisfy the schema constraints
                     then
                        equate s to r
                        replace i with iθ
                        for each support b and effect b of i do
                           replace b with bθ
                     else ProcessFact(s)
               for each effect e of x do
                  let d = Retrieve(¬e)
                  if d then Delete(d)
                  ProcessFact(e)
               for each super b of action i in the action hierarchy do
                  ProcessAction(b)
```

Figure 10.5: The Understanding Procedure

information upon which we base our prediction that an individual will have a certain goal (p. 132)." *Thematic goals* are therefore defined as goals that arise from basic human wants and needs and therefore require no further explanation. They are further classified as either *satisfaction goals* such as satisfying hunger or sexual desires, *achievement goals* such as acquiring money or power, *enjoyment goals* such as wanting to be entertained, and *preservation goals* such as preserving one's health or safety. GENESIS has a set of inference rules that determine when a thematic goal has been achieved. These inference rules effectively define a set of known thematic goals. For example, the following inference rule defines the thematic goal of wanting to possess valuable items:

Possess(?x,?y) $\wedge$ Valuable(?y) $\rightarrow$ ThemeGoalMet(?x, Possess(?x,?y))

The complete list of rules for inferring that a thematic goal has been achieved is given in section 2 of appendix C.

The outer loop of the understanding procedure in Figure 10.5 sequentially processes each of the inputs in the narrative. If an input is not an action and cannot be inferred from what is already known, then it is first added to the causal model. Next, suggested schemata are checked to see if the input matches a fact in one of their expansions. Finally, the system tries to infer that the fact achieves a thematic goal for some character. If it does, the explanation for how the goal was achieved is passed to the schema learner for possible generalization. In the murder example, all but the last three inputs are facts that simply define the various objects in the narrative and the relationships between them. All of these inputs are simply asserted in the causal model.

If an input is an action, it is first checked to see if it matches an action in the expansion of a suggested schema. If it does, this schema is a likely explanation for the action and the system proceeds to the next input. Otherwise, the system checks if it suggests a schema that might explain it. However, if an action achieves a thematic goal, it is not used to suggest schemata since the system already has an explanation for why the action was performed and does not need to search for an explanation in terms of it being part of a larger schema. An action is said to achieve a thematic goal if one of its effects can be inferred as satisfying a thematic goal of the actor. This process of preventing explained actions from suggesting schemata was first discussed in [Mooney85b] but was not implemented in the original version of GENESIS. The importance of this process is illustrated with an example in section B.2.4 of appendix B.

If an action does not suggest a schema that can explain it, then the system attempts to connect its preconditions to the effects of previous actions. As in the plan verification procedure given in chapter 6, the system tries to infer each of the supports of the action using the deductive retriever and if it fails, it simply assumes the support must be true and adds it to the database. An exception to this rule is that if a constraint contains an unbound schema variable, then the system does not try to infer that constraint. This is to prevent premature assignment of unknown variables. For example, if a particular role in the action is not mentioned and this role has a constraint that it be a Person, using the deductive retriever to satisfy this constraint could assign any known person to this role. If the support is a precondition or motivation with variables, then any schema variable bindings that are created by matching it to a known fact must satisfy the schema constraints.

After an action's preconditions have been considered, its effects are asserted and deletions are determined as in the procedure in section 6.2.1. The procedure ProcessFact is used to assert each effect and check if this state satisfies a thematic goal for some character. The final step in processing an action is to call ProcessAction recursively on any abstractions of the input action in the action hierarchy.

In the murder example, Agrippinna giving Claudius the poisonous mushroom and Claudius dying is explained in terms of Poison and Murder as described in section 10.1.2. When an action schema is determined, the procedure ProcessAction is called on the action in an attempt to connect it to previous actions. The following facts are supports of the Posion action:

Isa(Mushroom1, Food), Poisonous(Mushroom1),
NeedSustenance(Person1), Possess(Person2, Mushroom1)

The first one is inferred from the input: Isa(Mushroom1, Mushroom) using the object hierarchy. The second directly matches a previous input. The facts that Claudius was hungry and that Agrippina had the mushroom cannot be retrieved and are assumed to be true and asserted in the database. The following facts are asserted as effects of the Poison:

Inside(Mushroom1,Person1), ¬Possess(Person1, Mushroom1), ¬Possess(Person2, Mushroom1)

The final fact deletes the precondition that was just assumed. When ProcessAction is called on the determined Murder action, the only supports are the constraints:

Isa(Person1, Person), Isa(Person2, Character)

both of which are retrieved from input facts. The only fact added as an effect is: Dead(Person1).

The final input in the example is: Inherit(Person2, Estate1, ?FROM3), which states that Agrippina inherited the estate from someone. This does not match any action in a suggested schema nor does it suggest any schemata. Therefore, the system tries to connect it to previous actions. The supports of the action are:

Dead(?FROM3), Possess(?FROM3, Estate1), Heir(Person2, ?FROM3),

Isa(?FROM3, Person), Isa(Person2, Character), Isa(Estate1, Inanimate)

The first support matches the effect of the Murder: Dead(Person1) by binding the variable ?FROM3 to Person1. Since Person1 is a Person, this binding satisfies the schema constraint: Isa(?FROM3, Person) and this identification is made throughout the action. The presence of ?FROM3 in the supports also illustrates why constraints with variables are not allowed to be retrieved. If the support: Isa(?FROM3, Person) were considered first, ?FROM3 could be incorrectly bound to Person2 since Isa(Person2, Person) is in the database. Preconditions are generally less likely to result in such mistaken assignments. A still better approach might be to find a set of bindings that maximize the number of supports that can be retrieved; however, such a process could require searching an exponential number of possible bindings. Once ?FROM3 is bound to Person1, all of the constraints are easily deduced using the object hierarchy and the precondition Possess(Person1, Estate1) matches an input. The remaining precondition: Heir(Person2, Person1) is deduced from the input: Husband(Person1, Person2) using the following inference rules:

Husband(?x, ?y) → Spouse(?x, ?y)

Spouse(?x, ?y) → Spouse(?y, ?x)

Spouse(?x, ?y) → Heir(?x, ?y)

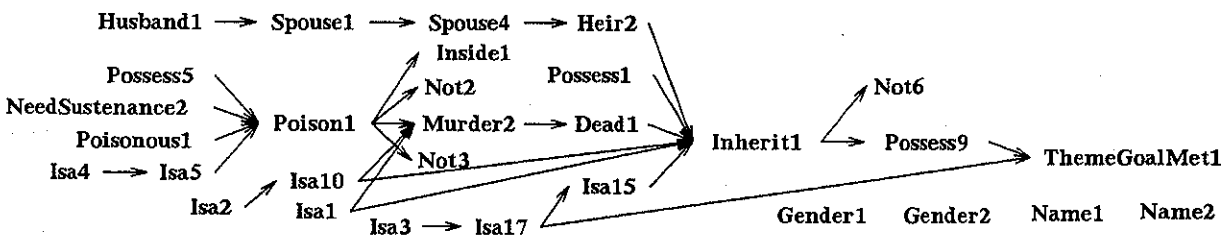
Finally, the following effects of the Inherit are asserted:

Possess(Person2, Estate1), ¬Possess(Person1, Estate1)

The second effect deletes the fact that Claudius owned the estate. The first effect is recognized as achieving a thematic goal for Agrippina since the object hierarchy specifies that an estate is a valuable item. The explanation for how this goal was achieved is shown in Figure 10.6. The explanation shown in the figure is called a *highest-level* explanation since it includes only the "largest" or highest-level schemata that support the goal. Actions that only support the goal because they are a part of the expansion of a higher-level schema are not included in the highest-level explanation. Consequently, the expansions of the Murder and Poison schemata are not included in Figure 10.6.

The explanation constructed by the understanding process allows the system to answer a number of questions about the narrative. Below is the input/output trace produced by GENESIS when it is run on this example:

Input: Claudius was Agrippina's husband and owned an estate. Agrippina gave him a poisonous mushroom and he died. She inherited the estate.



ThemeGoalMet1	ThemeGoalMet(Person2, Possess(Person2,Estate1))	Agrippina is happy that Agrippina has the estate.
Isa17	Isa(Estate1,Valuable)	The estate is a valuable.
Isa3	Isa(Estate1,Estate)	The estate is an estate.
Possess9	Possess(Person2,Estate1)	Agrippina has the estate.
Inherit1	Inherit(Person2,Estate1,Person1)	Agrippina inherited Claudius's estate.
Isa15	Isa(Estate1,Inanimate)	The estate is an inanimate object.
Isa1	Isa(Person1,Person)	Claudius is a person.
Isa10	Isa(Person2,Character)	Agrippina is a character.
Isa2	Isa(Person2,Person)	Agrippina is a person.
Dead1	Dead(Person1)	Claudius is dead.
Murder2	Murder(Person2,Person1)	Agrippina murdered Claudius.
Poison1	Poison(Person2,Person1,Mushroom1)	Agrippina poisoned Claudius with the mushroom.
Isa5	Isa(Mushroom1,Food)	The mushroom is a food.
Isa4	Isa(Mushroom1,Mushroom)	The mushroom is a mushroom.
Poisonous1	Poisonous(Mushroom1)	The mushroom is poisonous.
NeedSustenance2	NeedSustenance(Person1)	Claudius is hungry.
Possess5	Possess(Person2,Mushroom1)	Agrippina had the mushroom.
Inside1	Inside(Mushroom1,Person1)	The mushroom is inside Claudius.
Not2	¬Possess(Person2,Mushroom1)	Agrippina does not have the mushroom.
Not3	¬Possess(Person1,Mushroom1)	Claudius does not have the mushroom.
Possess1	Possess(Person1,Estate1)	Claudius had the estate.
Heir2	Heir(Person2,Person1)	Agrippina is Claudius's heir.
Spouse4	Spouse(Person2,Person1)	Agrippina is Claudius's spouse.
Spouse1	Spouse(Person1,Person2)	Claudius is Agrippina's spouse.
Husband1	Husband(Person1,Person2)	Claudius is Agrippina's husband.
Not6	¬Possess(Person1,Estate1)	Claudius does not have the estate.
Gender1	Gender(Person1,male)	Claudius is male.
Gender2	Gender(Person2,female)	Agrippina is female.
Name1	Name(Person1,Claudius)	Claudius's name is Claudius.
Name2	Name(Person2,Agrippina)	Agrippina's name is Agrippina.

Figure 10.6: Murder Example – Specific Explanation (Highest-Level)

Thematic goal achieved: Agrippina is happy that Agrippina has the estate.
Explanation suitable for generalization. Pruning...Generalizing...Packaging...
Creating New Schema: (MurderInherit ?a5 ?o4 ?d1)

?a5 is a character. ?d1 is a person. ?a5 murders ?d1. ?o4 is an inanimate object. ?o4 is a valuable. ?d1 has ?o4. ?a5 is ?d1's heir. ?a5 inherits ?d1's ?o4.

Having: ?a5 murdered ?d1. suggest MurderInherit

Ready for questions:

>Summarize
Agrippina murdered Claudius. Claudius had an estate. Agrippina was Claudius's heir. Agrippina inherited Claudius's estate.

>Why did Agrippina give Claudius the mushroom?
So Claudius would eat the mushroom and so Claudius would die.

>Why did Claudius eat the mushroom?
Because Claudius was hungry.

>Why is Claudius dead?
Because Agrippina poisoned Claudius with the mushroom.

>Why did Agrippina kill Claudius?
So Agrippina could inherit Claudius's estate.

>Why did Agrippina inherit the estate?
Because Claudius was Agrippina's husband, because Claudius had the estate and because Agrippina poisoned Claudius with the mushroom.

>Why is Agrippina Claudius's heir?
Because Claudius was Agrippina's husband.

10.3. Limiting Search in Understanding

GENESIS's understanding process is built on the assumption that combinatorially explosive search through possible explanations should be avoided. PAM [Wilensky78], on the other hand, is an explanation-based understanding system that searches for an explanation of an action until it exhausts its applicable knowledge. PAM searches for an explanation of a character's action by predicting future actions the character might take and seeing if any of them satisfy a known goal for that character. In terms of the representation presented here, when PAM processes an action, it searches through possible next actions that have preconditions that match the effects of the previous action. For example, consider the following narrative, which is normally used as an example of how PAM processes a narrative.

Willa was hungry. She picked up the Michelin guide and got in her car.

When the system processes the action that Willa has picked up the Michelin guide, it considers that the next action she might perform is to read the guide since possessing a book is a precondition for reading it. The effect of reading the Michelin guide is that she will know the location of a restaurant. Since knowing the location of a restaurant is a precondition for going to a restaurant, it considers the next action she might take is to go to a restaurant. Finally, since being at a restaurant is a precondition for eating there, it considers eating at the restaurant to be a possible next action. This action is realized as satisfying her hunger and therefore the chain of actions just discussed is considered to be the explanation for her actions. If at some time the system cannot propose a possible next action due to lack of knowledge of an action that has a precondition that matches the previous action's effect, then it backtracks to the last choice and picks another possible action. This search is continued until the system encounters an action that satisfies a known goal or until it has exhausted its knowledge and cannot predict any more possible future actions.

However, searching through a space of possible future actions is combinatorially explosive. Consequently, such an approach is intractable if a system's knowledge of actions is large, which it obviously must be if it is to be able to understand a wide range of narratives. Consider what a PAM-like system would do if it where used to process the murder-for-inheritance narrative. Upon determining that Agrippina had murdered Claudius, it would conduct an exhaustive search for an explanation of this action. Such a system would spend a long time exploring possible courses of action that Agrippina could take now that Claudius is dead. She could bury him, she could cremate him, she could donate his body to medical science, etc.. A system with a large knowledge base of actions could search a long time before stumbling upon the idea of her inheriting his property.

Currently, in order to avoid such combinatorially explosive search, GENESIS does not try to predict future (or past) courses of action in this manner. If an action does not suggest a known schema, the system only tries to connect its preconditions to the effects of a previous action. Of course, suggesting schemata can be seen as predicting possible future or past actions. In fact, if suggested schemata were immediately used to recursively suggest other schemata, the system would simply be conducting combinatorially explosive search through a different space.[5] However, in GENESIS, a suggested schema is not used to suggest other schemata until it has accounted for additional actions and is determined. Therefore, GENESIS' conservative approach to schema selection greatly limits the search it performs during this process.

Overall, GENESIS performs relatively little search during understanding. Its approach to understanding is for the most part *non-predictive* [Segre87a]. or "wait and see" [Marcus74]. In fact, GENESIS is a little *too* conservative with regards to searching for an explanation. Performing search for missing actions is clearly necessary for understanding narratives such as the PAM example shown above. But searching for missing actions must be tightly constrained since it quickly becomes intractable.

Since GENESIS does not conduct a complete search for an explanation, it is incapable of "understanding" narratives that have missing actions and do not suggest known schemata. For example, consider the processing of the following test narrative for the murder example *before* the system learns a murder-for-inheritance schema.

Input: Martha is Gene's mother and is a millionaire. He shot her and she died. Gene got $1000000.

Ready for questions:

>Summarize

Gene shot Martha and killed her.

[5] Plan-based understanding as performed by PAM can be modelled as schema selection if each action has an additional subaction for achieving each of its preconditions. In this case, if an action A meets a precondition for action B, then A must be be a subaction of B and therefore B would be suggested as a possible reason for performing A. If suggested schemata recursively suggest other schemata, the result is the same search through future actions that PAM performs. A schema is selected if it is the final action in a chain of actions that begins with the observed action and ends in the achievement of a thematic goal. The formulation of *motivation analysis* presented in [Charniak85 (chapter 10)] can be viewed as taking this approach.

>Why did Gene shoot Martha?

So Martha would die.

>Why did Gene kill Martha?

Cannot find sufficient reason.

>Why did Martha die?

Because Gene shot Martha and killed her.

>How did Gene get the money?

Cannot find sufficient cause.

The parsed representation of this narrative is shown in Figure 10.7. The crucial difference between this story and the Claudius and Agrippina story is that the current narrative only states that Gene has the money. It is missing the action declaring that he inherited it. A couple of additional actions used in the processing of this narrative are Shoot and ShootToKill (a specialization of Murder). Definitions for these actions are shown in Table 10.2 and Table 10.3.

When processing this story, the input: Shoot(Person2, Person1, ?INSTR6) suggests the schema: ShootToKill(Person2, Person1, ?INSTR6). The subsequent input is Die(Person1), which matches a

Isa(Person1,Person), Gender(Person1,female), Name(Person1,Martha),
Isa(Person2,Person), Gender(Person2,male), Name(Person2,Gene), Isa(Money1,Money),
Amount(Money1,order-millions,dollar), Mother(Person1,Person2), Possess(Person1,Money1),
Shoot(Person2,Person1,?INSTR6), Die(Person1), Isa(Money2,Money),
Amount(Money2,1000000,dollar), Possess(Person2,Money2)

Figure 10.7: Parsed Version of the Murder Test Narrative

Table 10.2: Additional Definitions of Action Supports and Effects

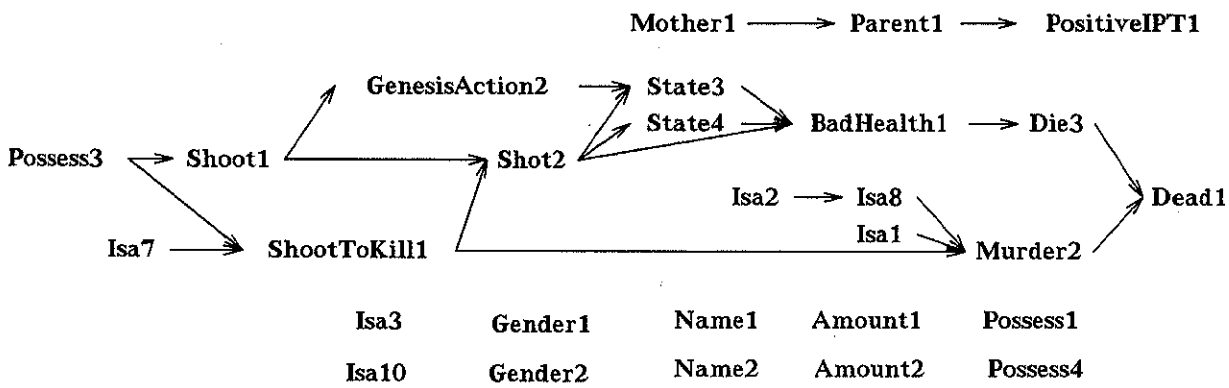
Action	Constraints	Preconditions	Effects
Shoot(?a,?v,?g) "?a shot ?v with ?g"	Isa(?g,Gun) Isa(?a,Person)	Possess(?a,?g) PointingAt(?a,?g,?v)	Shot(?v)
ShootToKill(?a,?v,?g) "?a shot ?v and killed it"	Isa(?g,Gun)	Possess(?a,?g)	Shot(?v)

Table 10.3: Expansion of ShootToKill

Subactions	Internals	Links
Shoot(?a,?v,?g) Die(?v)	BadHealth(?v)	(Precondition (Subaction 1) (Precondition 1)) (Effect (Subaction 1) (Effect 1)) (Antecedent (Internal 1) (Effect 1)) (Precondition (Subaction 2) (Internal 1)))
Abstractions		
((Effect 1) Murder (Internal 1)) ((Internal 1) Murder (Internal 3)) ((Subaction 1) Murder (Subaction 1))))		((Effect 1) Murder (Internal 2)) ((Subaction 2) Murder (Subaction 2))

subaction in the suggested ShootToKill schema. Since both of the subactions of the suggested schema have been explicitly mentioned, it is determined and added to the causal model. The facts that ?INSTR6 is a gun and that Gene possesses it are assumed since they are supports of the determined schema. The final input stating that Gene has $1,000,000 does not match part of a suggested schema and is simply added to the causal model as a disconnected fact. The final causal model constructed for this narrative is shown in Figure 10.8.

The causal connections added by the ShootToKill schema allows the system to answer a couple of questions about the narrative; however, the causal model is not sufficient to explain why Gene killed Mary or how he got the money. Without a schema for murder-for-inheritance, the system cannot infer the missing Inherit action since it does not conduct a potentially expensive search for an action that would connect Gene's acquisition to the rest of the narrative.



Dead1	Dead(Person1)	Martha is dead.
Murder2	Murder(Person2,Person1)	Gene murdered Martha.
ShootToKill1	ShootToKill(Person2,Person1,?INSTR6)	Gene shot Martha and killed her.
Isa7	Isa(?INSTR6,Gun)	The gun is a gun.
Possess3	Possess(Person2,?INSTR6)	Gene has the gun.
Shot2	Shot(Person1)	Martha is shot.
Shoot1	Shoot(Person2,Person1,?INSTR6)	Gene shot Martha with the gun.
Isa1	Isa(Person1,Person)	Martha is a person.
Isa8	Isa(Person2,Character)	Gene is a character.
Isa2	Isa(Person2,Person)	Gene is a person.
Die3	Die(Person1)	Martha died.
BadHealth1	BadHealth(Person1)	Martha is in bad health.
State4	State(Person1)	Martha is in some state.
State3	State(Person1)	Martha is in some state.
GenesisAction2	GenesisAction(Person2)	Gene did something.
PositiveIPT1	PositiveIPT(Person1,Person2)	Martha has a positive relationship with Gene.
Parent1	Parent(Person1,Person2)	Martha is Gene's parent.
Mother1	Mother(Person1,Person2)	Martha is Gene's mother.
Isa3	Isa(Money1,Money)	Millions of dollars is money.
Isa10	Isa(Money2,Money)	The $1000000 is money.
Gender1	Gender(Person1,female)	Martha is female.
Gender2	Gender(Person2,male)	Gene is male.
Name1	Name(Person1,Martha)	Martha's name is Martha.
Name2	Name(Person2,Gene)	Gene's name is Gene.
Amount1	Amount(Money1,order-millions,dollar)	Millions of dollars is order-millions dollars of money.
Amount2	Amount(Money2,1000000,dollar)	The $1000000 is 1000000 dollars of money.
Possess1	Possess(Person1,Money1)	Martha has millions of dollars.
Possess4	Possess(Person2,Money2)	Gene has the $1000000.

Figure 10.8: Causal Model for Murder Test Example Before Learning

CHAPTER 11

GENESIS: SCHEMA ACQUISITION

If the understander is capable of constructing an explanation for how a goal was achieved, GENESIS may be able to use EGGS to generalize this explanation and learn a new schema which can aid in the understanding of future narratives. First, the system must decide whether or not an explanation is worth generalizing. This process is described in section 11.1. Next, a special pruning procedure for EGGS eliminates overly specific parts of the explanation. The pruner used in GENESIS is described in section 11.2. Next, the EGGS generalizer is used to generalize the resulting explanation. In addition to this process, a couple of additional constraints must be added to insure that all actions are motivated. This process is discussed in section 11.3. Finally, a special packager for EGGS constructs a new schema from the generalized explanation and indexes the schema so that its subactions suggest it. This packager is an enhancement of the procedure for extracting partially-ordered macro-operators (see section 6.2.2) and is discussed in section 11.4. In addition, a description of how a learned schema can aid the understanding of future narratives is presented in section 11.5. Throughout the chapter, the learning of the murder-for-inheritance schema is used as an example.

11.1. Deciding When to Learn

If every explanation GENESIS encountered was generalized into a new schema, the system would soon become overloaded with rarely used schemata. Most actions would suggest a large number of schemata and selecting among these would require an excessive amount of processing time. This problem is analogous to the phenomenon of slowdown in problem solving systems that learn macro-rules (see section 7.2).

One approach to this problem is to generalize all explanations and later eliminate those schemata that are empirically found to be infrequently used. In the PEBLS model [Segre87a], the conditions used to determine which schemata are retained are called the *retention criteria*. An example of this approach is the learning of S-Macros in MORRIS [Minton85], which was discussed in chapter 7. Another approach is to selectively learn those schemata that are judged likely to be useful in latter processing. In the PEBLS model, the conditions used to determine whether or not an explanation is worth generalizing are called the *learning criteria*. The learning of T-Macros in MORRIS, also mentioned in chapter 7, is an example of this approach. Of course, these two approaches to determining utility are not necessarily mutually exclusive and MORRIS and PRODIGY [Minton87a, Minton87b] employ both techniques.

Currently, GENESIS only has criteria for selectively learning schemata. Below are the set of criteria that GENESIS uses to determine whether an explanation is worth generalizing.

(1) It should be an explanation of how a thematic goal was achieved.

(2) The highest-level explanation for the goal achievement should not be simply an instantiation of a known schema.

(3) All actions in the highest-level explanation should rely on the character whose thematic goal was achieved.

The first criterion is crucial for insuring that the acquired schema will be a useful one. A method for achieving a state that satisfies normal human wants and desires is likely to make a schema that will arise again and again. Recall that thematic goals were defined as the highest level goals that motivate a character's action. If an action achieves a thematic goal for its actor, it requires no further explanation. Consequently, an explanation is considered to be worth generalizing into a new schema only if it achieves a thematic goal. Since the understander detects the achievement of thematic goals and passes them on to the schema learner (see section 10.2), the schema learner itself does not need to explicitly check this criterion. In the murder example, the explanation shown in Figure 10.6 satisfies this criterion since it is an explanation of how Agrippina achieved her thematic goal of possessing a valuable item, namely, Claudius' estate.

The second criterion is the obvious one of not already possessing a schema for the combination of actions that achieves the goal. This simply involves checking the highest-level explanation for the goal achievement to make sure it contains several different actions. If the system already had a schema for this case, it would have used it in processing the narrative and the goal achievement would be explained at the highest level by an instance of this schema instead of by a combination of several actions. In the murder example, the highest-level explanation satisfies this criterion since it is composed of a Poison and an Inherit action. It should be noted that actions like Murder, which are generalizations of other actions in the explanation, do not count as separate actions.

The third and final criterion insures that the schema that is learned from the explanation is a volitional action that a character can use to achieve his own thematic goal. Let the term *main character* refer to the character whose thematic goal was achieved. If the learned schema is to represent a plan that can be executed by the main character to achieve his thematic goal, actions in the schema that are not volitionally performed by the main character should at least be motivated by actions that he performs. In order to insure this, non-volitional actions in the explanation must have a precondition supported by a volitional action performed by the main character. For example, the explanation for the murder story contains the non-volitional action Inherit; however, the precondition of this action that states that the benefactor must be dead is achieved by Agrippina's Murder action and Agrippina is the main character since it is her thematic goal that is achieved. Also, volitional actions in the highest-level explanation performed by actors other than the main character must have a motivation supported by an action performed by the main character. For example, in the learning of an arson-for-insurance schema (an example presented in appendix B), someone burns his own warehouse, which causes his insurance company to reimburse him for the loss. The explanation for this example can be generalized into a volitional schema. Even though the volitional Indemnify action in the explanation is performed by another actor (i.e. Prudential), this action is motivated by the fact that they believe his barn is burnt, which in turn is supported by the Burn action performed by the main character (see section *B.2.3 for details).

If the third criterion were not used, the system could learn schemata that contain serendipitous actions over which the main character has no control. For example, assume GENESIS had the knowledge to explain the following narrative but did not have the third learning criterion.

John's rich uncle was killed in an earthquake. John inherited a million dollars.

Such a system would acquire an EarthquakeInherit schema (or possibly a NaturalDisasterInherit schema) from its explanation of this narrative. Such a non-volitional schema would probably not be that useful for understanding later narratives.

11.2. Pruning the Explanation

If an explanation meets all three learning criteria, GENESIS proceeds to generalize it into a new schema. In order to increase to the generality and applicability of the resulting schema, GENESIS has a pruning procedure that removes unnecessarily specific branches from the explanation prior to generalization. If such pruning were not performed, the system would frequently learn schemata that are too restrictive to be useful. For example, if the explanation for the Claudius and Agrippina narrative were not pruned, the resulting schema would only cover cases in which a wife poisoned her husband with a mushroom in order to inherit an estate.

A description of the GENESIS pruning algorithm is given in Figure 11.1. The fact that only the highest-level explanation is considered during generalization is in itself a type of pruning. This

for each inference rule $a_1...a_n \rightarrow c$ in the highest-level explanation structure **do**
  if PrunableType($a_1...a_n \rightarrow c$)
    then
      let q = true
      for each fact f which eventually supports c **do**
        for each path p from f to the goal **do**
          let r = true
          for each inference rule $b_1...b_m \rightarrow d$ in p **do**
            if PrunableType($b_1...b_m \rightarrow d$) **then let** let r = false
          if r **then let** q = false
      if q **then** remove $a_1...a_n \rightarrow c$ from explanation

function PrunableType($a_1...a_n \rightarrow c$)
  if $a_1...a_n \rightarrow c$ is an abstraction inference **then return** true
  if there is more than one inference rule in the library whose conclusion unifies with c
    then return true
    else return false

Figure 11.1: GENESIS Pruning Procedure

is because the highest-level explanation does not include expansions of schemata that only support the goal through the overall effects of the schema. For example, in the murder example, the highest-level explanation (see Figure 10.6) does not include the Atrans, Ingest, or Die actions in the expansion of the Poison schema.

In addition, the procedure in Figure 11.1 attempts to prune certain types of inferences from the highest-level explanation structure. One type of *prunable* inference includes abstraction inferences based on GENESIS' hierarchies of objects, states, or actions. For example, below is a list of the abstraction inferences in the explanation of the Claudius and Agrippina example:

Isa(Mushroom1, Mushroom) → Isa(Mushroom, Food)

Isa(Estate1, Estate) → Isa(Estate1, Valuable)

Isa(Estate1, Valuable) → Isa(Estate1, Inanimate)

Isa(Person2, Person) → Isa(Person2, Character)

Husband(Person1, Person2) → Spouse(Person1, Person2)

Poison(Person2, Person1, Mushroom1) → Murder(Person2, Person1)

Another type of prunable inference includes inferences that conclude facts that could be deduced in a variety of other ways in general. For example, below is a list of such inferences in the murder example.

Spouse(Person1, Person2) → Spouse(Person2, Person1)

Spouse(Person2, Person1) → Heir(Person1, Person2)

The first rule is in this class because the system has rules for deducing that someone is somebody's spouse other than by commutativity (e.g. from Husband, Wife, or Married). The second rule is in this class because the system has rules for deducing that someone is somebody's heir other than from an instance of Spouse (e.g. from an instance of Parent). In order to prevent the system from generating a schema that has a support that is logically entailed by another support, not all prunable inferences are immediately removed from the explanation. An inference rule that is classified as prunable is not pruned if it is supported by a fact that cannot itself be pruned because it supports the goal directly (i.e. not through a prunable inference). In the murder example, the rule:

Isa(Estate1, Valuable) → Isa(Estate1, Inanimate)

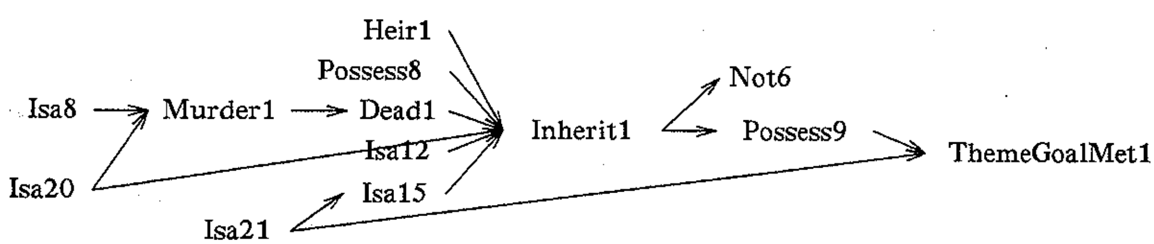
is not actually pruned because the antecedent: Isa(Estate1, Valuable) directly supports the fact that a thematic goal has been achieved. If this rule were pruned, the resulting schema would have two constraints on the inherited item, one stating that it is a Valuable and one stating that it is Inanimate since both facts would become leaves of the final explanation. However, the latter fact can be easily deduced from the former, as it was in the original explanation. Retaining prunable rules that are supported by facts that directly support the goal prevents such redundant constraints from appearing in the final schema. All of the other prunable rules listed above are removed since they are not supported by facts that support the goal through a path that does not contain a prunable inference.

Not pruning rules with antecedents that directly support the goal also prevents the removal of specialized actions that are crucial to the goal achievement. For example, if the Poison action had an effect that was not inherited from Murder and that supported the goal achievement directly, then it would not have been pruned. However, since all of its effects that eventually support the goal are inherited from Murder, the system does not consider it to be crucial to the overall plan and removes it.

Due to the representation of actions and their abstractions, the pruning algorithm shown in Figure 11.1 is capable of performing the task of *generalizing predicates* discussed in [DeJong86b]. Generalizing predicates refers to the process of altering the descriptions of actions by changing relatively specific predicates such as Poison to more general predicates such as Murder. Unlike the current version, the original GENESIS system [Mooney85a, Mooney85b], required a special procedure for this task.

11.3. Generalizing the Explanation

After the initial explanation is pruned, it is generalized using the EGGS generalizer. The generalized explanation for the murder example is shown in Figure 11.2. Unfortunately, this generalization does not correspond to the intuitive generalization of the Claudius and Agrippina narrative since the explanation structure itself does not impose the constraint that the murderer (?x26) and the heir (?a5) be the same. This is because the schema still "works" if they are different; however, the Murder is left unmotivated in this case. Since the resulting plan contains an unmotivated action by another agent, it does not represent a volitional action and is probably not worth learning. However, the plan can be made volitional by requiring the actor of every unmotivated



ThemeGoalMet1	ThemeGoalMet(?a5,Possess(?a5,?o4))	?a5 is happy that ?a5 has ?o4.
Isa21	Isa(?o4,Valuable)	?o4 is a valuable.
Possess9	Possess(?a5,?o4)	?a5 has ?o4.
Inherit1	Inherit(?a5,?o4,?d1)	?a5 inherited ?d1's ?o4.
Isa15	Isa(?o4,Inanimate)	?o4 is an inanimate object.
Isa20	Isa(?d1,Person)	?d1 is a person.
Isa12	Isa(?a5,Character)	?a5 is a character.
Dead1	Dead(?d1)	?d1 is dead.
Murder1	Murder(?x26,?d1)	?x26 murdered ?d1.
Isa8	Isa(?x26,Character)	?x26 is a character.
Possess8	Possess(?d1,?o4)	?d1 had ?o4.
Heir1	Heir(?a5,?d1)	?a5 is ?d1's heir.
Not6	¬Possess(?d1,?o4)	?d1 does not have ?o4.

Figure 11.2: Murder Example -- Initial Generalized Explanation

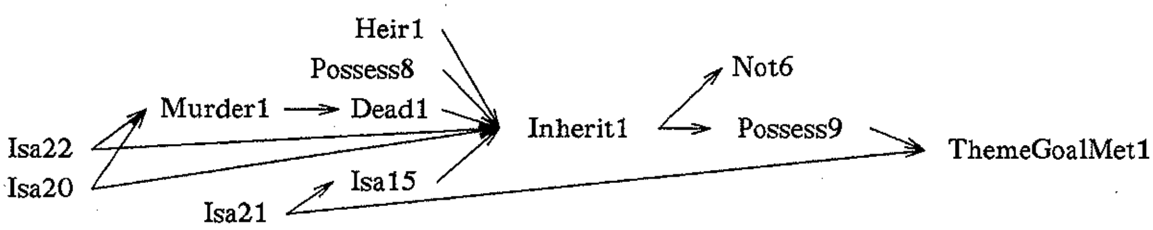
volitional action to be the same as the character whose thematic goal is achieved. A more formal description of this process is given in Figure 11.3. This procedure insures that actions that are otherwise unmotivated are motivated by the eventual thematic goal achievement. Any volitional action in the explanation that was performed by another actor in the original instance is guaranteed to have motivation supports. This is because any unmotivated action by another actor would have violated the third learning criterion and prevented generalization. In the murder example, the procedure in Figure 11.3 substitutes ?a5 for ?x26 and produces the final generalization shown in Figure 11.4.

It might be argued that requiring an additional step to fix such a problem is somewhat *ad hoc* and that all constraints should be represented in the explanation structure and enforced by the normal generalization process. A better solution to this problem might be to explicitly add motivations to actions in the explanation structure. That is, if the ultimate goal is: ThemeGoalMet(x,y) the motivation: Goal(a_i,y) would have to be added to each unmotivated action, where a_i is the actor of that action in the explanation structure. All of these motivations would then have to be equated to the form: Goal(x,y) in order to insure that all of these actions were motivated by the final goal achievement. The normal EGGS process would then enforce the unifications now enforced by the procedure in Figure 11.3. Although this approach is in some sense "cleaner," it is functionally equivalent to the procedure in Figure 11.3, and the added motivations would have complicated the question-answering process.

At this point, a brief digression is warranted regarding the third learning criterion since it interacts in an interesting way with the process of equating actors to the main character. For

let m be the variable for the main character in the generalized explanation
for each volitional action a in the generalized explanation **do**
  if a does not have any motivation supports
    then
      let b be the actor of a
      if b$\neq$m **then**
        let $\theta = \{m/b\}$
        for each wff f in the generalized explanation **do**
          replace f **with** fθ

Figure 11.3: Procedure for Equating Actors to the Main Character



ThemeGoalMet1	ThemeGoalMet(?a5,Possess(?a5,?o4))	?a5 is happy that ?a5 has ?o4.
Isa21	Isa(?o4,Valuable)	?o4 is a valuable.
Possess9	Possess(?a5,?o4)	?a5 has ?o4.
Inherit1	Inherit(?a5,?o4,?d1)	?a5 inherited ?d1's ?o4.
Isa15	Isa(?o4,Inanimate)	?o4 is an inanimate object.
Isa20	Isa(?d1,Person)	?d1 is a person.
Isa22	Isa(?a5,Character)	?a5 is a character.
Dead1	Dead(?d1)	?d1 is dead.
Murder1	Murder(?a5,?d1)	?a5 murdered ?d1.
Possess8	Possess(?d1,?o4)	?d1 had ?o4.
Heir1	Heir(?a5,?d1)	?a5 is ?d1's heir.
Not6	¬Possess(?d1,?o4)	?d1 does not have ?o4.

Figure 11.4: Murder Example – Generalized Explanation

example, the third learning criterion prevents GENESIS from generalizing the explanation constructed for the following narrative.

Bob is Jane's husband and is a millionaire. Stan murdered Bob and Jane inherited $1,000,000.

This is because Jane's inheritance is not supported by her own volitional action. However, the generalization EGGS produces for the explanation of this example is exactly the same as the one it produces for the Claudius and Agrippina narrative (see Figure 11.2). If the process of equating the actors of otherwise unmotivated actions to the main character is also applied to the generalization of this narrative, the murderer is required to be the same as the heir and the result is the same as the final generalized explanation for the Claudius and Agrippina story (see Figure 11.4). GENESIS could be altered to apply the third learning criterion only *after* the explanation has been generalized, in which case it could learn the same schema from this story as it learns from the Claudius and Agrippina one. However, since in the general case this process would be equating actors that were not necessarily the same in the specific instance, it might be equating variables that result in a protection violation (see section 6.2.2.1). Consequently, judging the third learning criterion would have to be delayed even further until the packaging process determined the necessary constraints to prevent protection violations. If equating all the actors of unmotivated actions to the main character did not violate any of these constraints, then it could be carried out and a volitional schema could be learned.

This approach could result in learning a schema that is not a generalization of the narrative that gave rise to it. For example, the schema that would be acquired from the narrative about Bob and Jane would require that the murderer and the heir be the same although they were different in the original story. Since the goal of EBL is to maintain the structure of the explanation for the original example, this approach is not taken and the third learning criteria is applied prior to generalization. Also, waiting to apply the third criterion could result in wasting work on pruning, generalizing, and packaging an explanation that does not result in acquiring a schema since it fails to meet the learning criteria.

11.4. Packaging the Explanation

The final step in learning a schema is packaging the generalized explanation into a form suitable for the schema library. The packaging process used in GENESIS is a slight enhancement of the procedure for learning partially-ordered macro-operators (see section 6.2.2). First the leaves of the generalized explanation are divided into constraints, preconditions, and motivations based on the manner in which they support actions in the generalized explanation. For example, since the leaf: Heir(?a5, ?d1) is a precondition of the Inherit action in the generalized explanation for the murder example, it is made a precondition of the overall schema. Protection violations are prevented and the effects of the schema are determined using the procedure presented in section 6.2.2. The supports and effects for the schema constructed from the murder example are shown in Table 11.1. A name for the new schema, such as MurderInherit, is constructed by concatenating the names of the actions in the generalized explanation.

Table 11.1: Supports and Effects of the MurderInherit Schema

MurderInherit(?a5,?o4,?d1)		
Constraints	Preconditions	Effects
Isa(?a5,Character) Isa(?d1,Person) Isa(?o4,Valuable)	Heir(?a5,?d1) Possess(?d1,?o4)	Dead(?d1) ¬Possess(?d1,?o4) Possess(?a5,?o4)

In addition, the internal structure of the explanation is stored as the expansion of the new schema. The actions in the explanation form the subactions of the new schema and other facts in the explanation that are neither supports nor effects make up the internals. The connections between all of the facts in the explanation make up the links of the new schema. The expansion that GENESIS constructs for the MurderInherit schema is shown in Table 11.2. An English summary of the resulting schema is given in the trace for this example presented in section 10.2.

The final step in the packaging of a new schema is to index the schema so that the system is "reminded" of it whenever it might be helpful in processing a future narrative. Indexing is performed by having the schema's volitional subactions suggest it. This is appropriate because the new schema may now provide an explanation for why someone is executing the subaction. In the murder example, the following suggestion is added to the Murder action:

Murder(?a5, ?d1) $\rightsquigarrow$ MurderInherit(?a5, ?o4, ?d1)

A suggestion is not added to Inherit because it is a non-volitional action.

11.5. Using the Learned Schema

Once a schema is added to the schema library and suggestions are added to index it, instances of its subactions in future narratives will suggest it. If an instance of the new schema is subsequently determined, it can result in GENESIS inferring missing actions that it was previously incapable of inferring. For example, consider the processing of the murder test narrative after the system has acquired MurderInherit.

Input: Martha is Gene's mother and is a millionaire. He shot her and she died. Gene got $1000000.

Table 11.2: Expansion of MurderInherit

Subactions	Internals
Murder(?a5,?d1) Inherit(?a5,?o4,?d1)	Isa(?o4,Inanimate)
Links	
(Antecedent (Expansion 1) (Constraint 3)) (Constraint (Determine 1) (Constraint 1)) (Precondition (Determine 2) (Effect 1)) (Precondition (Determine 2) (Precondition 1)) (Constraint (Determine 2) (Constraint 2)) (Effect (Determine 2) (Effect 2))	(Constraint (Determine 1) (Constraint 2)) (Effect (Determine 1) (Effect 1)) (Precondition (Determine 2) (Precondition 2)) (Constraint (Determine 2) (Expansion 1)) (Constraint (Determine 2) (Constraint 1)) (Effect (Determine 2) (Effect 3))

Thematic goal achieved: Gene is happy that Gene has the $1000000.

Ready for questions:

>Summarize

Gene murdered Martha. Martha had $1000000. Gene was Martha's heir. Gene inherited Martha's $1000000.

>Why did Gene kill Martha?

So Gene could inherit Martha's $1000000.

>How did Martha die?

Gene shot Martha and killed her.

>How did Gene get the money?

Gene inherited Martha's $1000000.

>Why did Gene inherit the money?

Because Gene shot Martha and killed her, because Martha had millions of dollars and because Martha was Gene's mother.

>Why was Gene Martha's heir?

Because Martha was Gene's mother.

>Why did Martha have $1000000?

Because Martha had millions of dollars.

This time, when a Murder action is determined from the actions in the second sentence (see section 10.3), it suggests the following instance of the new schema:

MurderInherit(Person2, ?o4-2, Person1)

In addition, the Murder action in the suggested schema is marked "found" as are its following supports and effects:

Isa(Person2, Character)

Isa(Person1, Person)

Dead(Person1)

The final input: Possess(Person2, Money2) matches an effect of the Inherit action in the suggested MurderInherit schema and binds ?o4-2 to Money2 since Money2 satisfies the constraint of being a Valuable. Since both of the positive effects of the Inherit action are now marked as found (i.e. Dead(Person1) and Possess(Person2, Money2)), the MurderInherit schema is determined and the missing Inherit action is added to the causal model. The precondition stating that Martha must have possessed $1,000,000 is inferred from the fact that she had millions of dollars and the precondition stating that Gene is Martha's heir is inferred from the fact that she is his mother. The final explanation that GENESIS constructs for this example is shown in Figure 11.5. This explanation allows the system to answer questions that it could not answer based on the representation it built without the MurderInherit schema (see Figure 10.8). The MurderInherit schema also provides a structure for paraphrasing the narrative.

The MurderInherit schema can also be used to understand narratives missing even more information. For example, consider the following GENESIS' input/output trace:

Input: Mary is the Bob's mother and is a millionaire. Someone strangled Mary. Bob got $1000000.

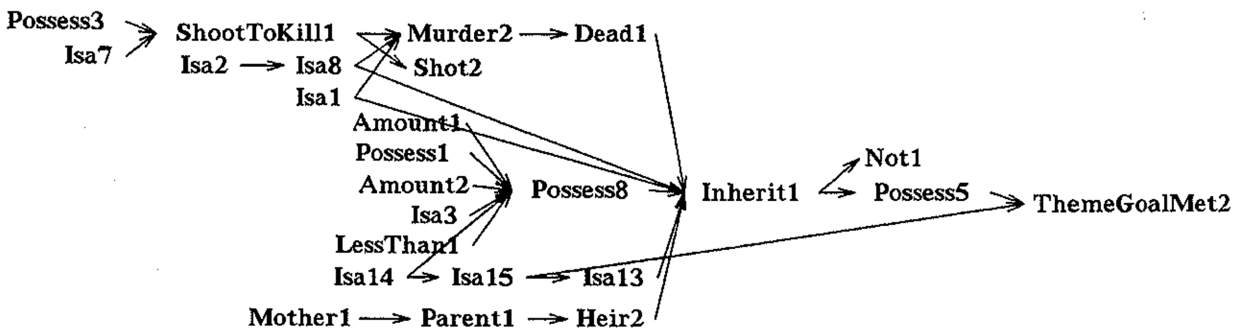
Thematic goal achieved: Bob is happy that Bob has the $1000000.

Ready for questions:

>Summarize
Bob murdered Mary. Mary had $1000000. Bob was Mary's heir. Bob inherited Mary's $1000000.

>How did Bob get the money?
Bob inherited Mary's $1000000.

>Who strangled Mary?
Bob strangled Mary.



ThemeGoalMet2	ThemeGoalMet(Person2, Possess(Person2,Money2))	Gene is happy that Gene has the $1000000.
Isa15	Isa(Money2,Valuable)	The $1000000 is a valuable.
Isa14	Isa(Money2,Money)	The $1000000 is money.
Possess5	Possess(Person2,Money2)	Gene has the $1000000.
Inherit1	Inherit(Person2,Money2,Person1)	Gene inherited Martha's $1000000.
Isa8	Isa(Person2,Character)	Gene is a character.
Isa2	Isa(Person2,Person)	Gene is a person.
Isa1	Isa(Person1,Person)	Martha is a person.
Isa13	Isa(Money2,Inanimate)	The $1000000 is an inanimate object.
Heir2	Heir(Person2,Person1)	Gene is Martha's heir.
Parent1	Parent(Person1,Person2)	Martha is Gene's parent.
Mother1	Mother(Person1,Person2)	Martha is Gene's mother.
Possess8	Possess(Person1,Money2)	Martha had the $1000000.
LessThan1	LessThan(1000000,order-millions)	1000000 is less than order-millions.
Isa3	Isa(Money1,Money)	Millions of dollars is money.
Amount2	Amount(Money2,1000000,dollar)	The $1000000 is 1000000 dollars of money.
Possess1	Possess(Person1,Money1)	Martha has millions of dollars.
Amount1	Amount(Money1,order-millions,dollar)	Millions of dollars is order-millions dollars of money.
Dead1	Dead(Person1)	Martha is dead.
Murder2	Murder(Person2,Person1)	Gene murdered Martha.
ShootToKill1	ShootToKill(Person2,Person1,?INSTR6)	Gene shot Martha and killed her.
Isa7	Isa(?INSTR6,Gun)	The gun is a gun.
Possess3	Possess(Person2,?INSTR6)	Gene has the gun.
Shot2	Shot(Person1)	Martha is shot.
Not1	¬Possess(Person1,Money2)	Martha does not have the $1000000.

Figure 11.5: Murder Test Example After Learning -- Specific Explanation

>Why did Bob strangle Mary?

So Bob could inherit Mary's $1000000.

Since Strangle is a specialization of Murder, the input: Strangle(?ACTOR1, Person1) suggests the schema: MurderInherit(?ACTOR1, ?o4-2, Person1). The input stating that Bob got a million dollars matches the effect: Possess(?ACTOR1, ?o4-2) of the suggested schema since Bob is a character and a million dollars is a valuable item. This match leads to the determination of MurderInherit and identifies Bob as the actor.

However, it is also important to note that not every narrative that might possibly fit the new schema actually suggests it. For example, consider the following GENESIS' input/output trace:

Input: Martha is Gene's mother and is a millionaire. She died. Gene inherited $1000000.

Thematic goal achieved: Gene is happy that Gene has the $1000000.

Ready for questions:

>Summarize
Martha died. Gene inherited Martha's $1000000.

>Who murdered Martha?
Question refers to unknown event, person, place, or thing.

>Why did Martha die?
Cannot find sufficient reason.

>Why did Gene inherit the money?
Because Martha was Gene's mother, because Martha had millions of dollars and because Martha died.

The system identifies Martha as the benefactor of the inheritance since she is the one that is dead and has the money; however, MurderInherit is never suggested because the Inherit action is non-volitional and does not suggest the new schema.

CHAPTER 12

GENESIS: LEARNING WORD MEANINGS

As revealed in the I/O trace presented in chapter 2, GENESIS also has the ability to learn provisional meanings for words from one example of their use. While processing the learning narrative for the kidnapping example, the system encounters the following unknown words: "kidnapper," "ransom," and "kidnap." Based on the context in which these words appear and their relation to the overall schema that is learned, the system acquires preliminary meanings for each of these unknown words. This chapter describes the procedure GENESIS uses to learn related word meanings when it is acquiring a new schema.[1] Section 12.1 attempts to motivate the approach taken in GENESIS by revealing a problem with previous approaches. Next, since the procedure for learning *role labels* such as "kidnapper" and "ransom" is significantly different from that used to learn *schema labels* such as "kidnap," a separate section is dedicated to each of these processes.

12.1. A Problem with Previous Models of Learning Word Meanings

Previous computational models of the acquisition of word meaning [Berwick83, Granger77, Selfridge82] have assumed existing knowledge of the concept underlying the word to be learned. In these models, word learning is a process of using surrounding context to establish an identification between a new lexical item and a known concept. However, new words are not always encountered as labels for known concepts. When encountering a new concept in natural language text or discourse, it is quite likely that one will also come across unknown words that refer to various aspects of the new concept. A word learning model that requires prior knowledge of the underlying concept will be unable to acquire even provisional meanings for such words. For example, such an approach could not learn meanings for the unknown words in the learning narrative for the kidnap example since when the system first encounters these words, it does not have any knowledge of the concepts underlying them.

In addition, developmental studies in psychology suggest that the learning of words and their underlying concepts frequently occurs concurrently in children. Experiments by Gopnik and Meltzoff revealed that children's acquisition of "disappearance" words occurred at about the same time they learned to solve object-permanence tasks involving invisible displacements [Gopnik86].

[1] A description of GENESIS' word learning abilities is also presented in [Mooney87a].

From this data, they concluded that learning may often involve "*concurrent* cognitive and semantic developments, rather than involving cognitive prerequisites for semantic developments" (p. 1051). They also state: "This raises the interesting possibility that conceptual and semantic development may occur concurrently, with each area of development influencing and facilitating the other.(p. 1051)" Bowerman [Bowerman80] and Kuczaj [Kuczaj82] have also used developmental data to argue for an interactive approach to language and concept acquisition.

In GENESIS, word learning is integrated with schema acquisition. This integration allows the system to simultaneously learn both a concept and a word that refers to the concept. Consequently, GENESIS can acquire definitions for words that cannot be learned by a system that requires prior conceptual knowledge. In addition, the system exhibits behavior similar to that observed in children's acquisition of word meanings.

12.2. Learning Role Labels

Role labels are words that refer to the role that an object or character plays in a schema. Examples of role labels include "kidnapper," "ransom," and "victim." The procedure GENESIS uses to learn role labels is similar to the technique used by the FOUL-UP system [Granger77] except that it is integrated with the schema learning and schema suggestion processes. When the parser encounters an unknown word where it expects a noun for an object of a particular class, a dummy variable is created, annotated with the unknown word, and allowed to fill the expectation. For example, the phrase "Fred paid him the ransom" in the kidnapping narrative is parsed into the assertion: Atrans(Person7, ?PhysicalObject1, Person9, ?AT24) where the variable ?PhysicalObject1 is marked with the fact that it came from the unknown word "ransom." If an input with an unknown-word variable like ?PhysicalObject1 matches a form in a suggested schema, then a provisional definition for the word is made based on the constraints on the schema variable that matches the unknown-word variable. For example, in the case of the unknown word "ransom," the previous sentence in the story suggests a Bargain schema between John and Fred. The subactions of this suggested schema are:

1) John told Fred that if Fred gave John the $250000 at Trenos then John would release Mary.
2) Fred gave John the $250000 at Trenos.
3) John released Mary.

Since "Fred paid him the ransom" matches the suggested subaction "Fred gave John the $250000 at Trenos" and since "ransom" fills the role of the item whose possession is transferred, an initial definition is made for "ransom" stating that it is a physical object whose possession is transferred during a Bargain.

However, this is not the final definition created for "ransom" since an additional process is performed when a new schema is learned. Each of the subactions composing a new schema is checked for roles that are filled by unknown-word variables or that were previously matched to such a variable resulting in an initial definition. In either case, a new definition is created for the unknown word based on the role it fills in the learned schema and the schema constraints on this role. Consequently, when the CaptureBargain schema is subsequently learned from the kidnapping narrative, it causes "ransom" to be redefined as a valuable item whose possession is transferred to the actor in a CaptureBargain schema. In other words, the word "ransom" is tentatively associated with the schema variable ?y5 in CaptureBargain, whose English paraphrase is repeated below for convenience:

> ?b9 is a person. ?c4 is a location. ?r5 is a room. ?c4 is in ?r5. ?x55 is a character. ?b9 is free. ?x55 captures ?b9 and locks him/her in ?r5. ?a34 is a character. ?x55 contacts ?a34 and tells it that ?b9 is ?x55's captive. ?y5 is a valuable. ?x55 wants to have ?y5 more than it wants ?b9 to be ?x55's captive. ?a34 has a positive relationship with ?b9. ?a34 has ?y5. ?x55 and ?a34 carry out a bargain in which ?x55 releases ?b9 and ?a34 gives ?x55 ?y5 at ?l11.

The rationale for having *learned* schemata take precedence when defining new role labels is that a learned schema represents a new situation and therefore new words are deemed more likely to be directly associated with it than with an existing schema like Bargain.

The provisional definition added to the lexicon for a role label contains two parts. The first is a set of constraints on the object itself, such as the constraint that "ransom" refers to a Valuable. The second part is a suggestion of the schema of which it is a role. The fact that a role label definition can suggest a relevant schema allows GENESIS to use the definition to construct explanations for narratives that it otherwise would not understand. For example, the definition acquired for the word "ransom" is crucial to the system's understanding of the following narrative.

> Input: Ted is Alice's husband. John took Alice into a room. Ted paid John the ransom and John released Alice.
>
> Thematic goal achieved: John is happy that John has the ransom.

Ready for questions:

>Summarize

Alice was free. John captured Alice and locked her in a room. John contacted Ted and told him that Alice was John's captive. John wanted to have a ransom more than he wanted Alice to be John's captive. Ted had a positive relationship with Alice. Ted had the ransom. John and Ted carried out a bargain in which John released Alice and Ted gave John the ransom.

>Why did John take Alice into the room?

So John could confine Alice in the room.

>Why did Ted pay John?

Because Ted believed that if Ted gave John the ransom then John would release Alice and because Ted wanted Alice to be free more than he wanted to have the ransom.

When the word "ransom" is encountered in this narrative, it suggests that the CaptureBargain schema might be relevant. This schema is then used in a top-down fashion to construct an explanation for the text. Since no other piece of information suggests CaptureBargain, the learned definition for "ransom" is crucial in understanding this story. If the definition for this word is removed, the system is no longer able to understand the narrative as illustrated by the following trace:

Input: Ted is Alice's husband. John took Alice into a room. Ted paid John the ransom and John released Alice.

Ready for questions:

>Summarize

John moved Alice into a room. Ted gave John a ransom. John released Alice.

>Why did John take Alice into the room?

Cannot find sufficient reason.

>Why did Ted pay John?

Cannot find sufficient reason.

A definition that associates the word "kidnapper" with the main character of a CaptureBargain schema (?x55) is learned in a manner directly analogous to the learning of the word "ransom." This word is also made to suggest the new schema, and this suggestion allows GENESIS to understand the following narrative:

Input: Ted is Alice's husband. A kidnapper took Alice into a room. Bob got 75000 dollars and released Alice.

Thematic goal achieved: Bob is happy that Bob has the $75000.

Ready for questions:

>Summarize
Alice was free. Bob captured Alice and locked her in a room. Bob contacted Ted and told him that Alice was Bob's captive. Bob wanted to have $75000 more than he wanted Alice to be Bob's captive. Ted had a positive relationship with Alice. Ted had the $75000. Bob and Ted carried out a bargain in which Bob released Alice and Ted gave Bob the $75000.

>Who took Alice into the room?
Bob moved Alice into the room.

>Why did Bob take Alice into the room?
So Bob could confine Alice in the room.

>How did Bob get the money?
Bob kidnapped Alice.

If the definition for "kidnapper" is removed, the following trace reveals the result:

Input: Ted is Alice's husband. A kidnapper took Alice into a room. Bob got 75000 dollars and released Alice.

Ready for questions:

>Summarize

A kidnapper moved Alice into a room. Bob released Alice.

>Who took Alice into the room?

The kidnapper moved Alice into the room.

>Why did Bob take Alice into the room?

Question refers to unknown event, person, place, or thing.

>How did Bob get the money?

Cannot find sufficient cause.

Although learning definitions for words such as "kidnapper" and "ransom." can be very helpful, many times a new word that fills a slot in a schema will not actually be a role label. For example, consider replacing the word "ransom" in the kidnap learning narrative with the word "moolah." Since the word "moolah" is unknown, GENESIS gives it a definition identical to the one it learns for "ransom." In order to be able to recover from such mistakes, the system monitors the schemata suggested by newly learned words. If a new word subsequently suggests a schema that does not explain any future inputs, the suggestion is removed. For example, consider processing the following murder-for-inheritance story after the system has acquired a "ransom definition" for "moolah."

Mary had $100000. Stan murdered Mary and inherited the moolah.

The word "moolah" suggests a CaptureBargain schema; however, none of the actions in the narrative match the subactions in its expansion. Consequently, the suggestion is considered to be misleading and is removed from the learned word.

12.3. Learning Schema Labels

Schema labels are verbs that refer to an entire plan schema. Examples of schema labels include "kidnap," "rob," and "poison." Learning meanings for schema labels is a more difficult task since the relevant context is potentially much broader. A sentence such as "John robbed the store" may be used to introduce a long piece of text elaborating the situation, to succinctly summarize a previous piece of text, or to refer to a single action in an even larger plan. A few heuristics have been developed that allow a reasonable guess to be made regarding the referent of such unknown verbs. The following one is used to resolve the meaning of "kidnap" as used in the kidnap learning narrative.

> If one character informs another that some unknown action occurred and a schema whose actor is the same as this action's was recently acquired from the narrative, and this schema also has roles filled by the speaker and any direct and indirect objects of the action, then assume that the speaker is summarizing the event and that the unknown act refers to the new schema.

Specifically, since Fred tells his wife that "someone kidnapped Mary" and since both he and Mary were participants in the just completed CaptureBargain schema, GENESIS assumes that the word "kidnap" refers to CaptureBargain. A definition for "kidnap" is added to the lexicon where it can be used in the parsing of future sentences. This definition states that the word refers to an instance of CaptureBargain in which the subject of the clause is the actor (?x55) and the direct object is the victim (?b9). Of course, the word "kidnap" is frequently used to refer to only an act of abduction rather than to an entire kidnap-for-ransom schema; however, the definition that GENESIS learns is a reasonable one given its current experience with the new word. In addition, a data-structure is added to the vocabulary of the generator where it can be used in the production of English replies to users' questions. For "kidnap", the added data-structure states that an instance of CaptureBargain can be stated in English by saying: "?x55 kidnapped ?b9."

The new definition of "kidnap" in the lexicon is crucial to GENESIS' understanding of the following narrative:

Input: Steve kidnapped Valerie. Mike was Valerie's father and paid Steve $30000.

Thematic goal achieved: Steve is happy that Steve has the valuable.

Ready for questions:

>Summarize

Valerie was free. Steve captured Valerie and locked her in a room. Steve contacted Mike and told him that Valerie was Steve's captive. Steve wanted to have $30000 more than he wanted Valerie to be Steve's captive. Mike had a positive relationship with Valerie. Mike had the

$30000. Steve and Mike carried out a bargain in which Steve released Valerie and Mike gave Steve the $30000.

>Why did Mike pay Steve the money?

Because Mike believed that if Mike gave Steve the $30000 then Steve would release Valerie and because Mike wanted Valerie to be free more than he wanted to have the $30000.

>Why did Steve kidnap Valerie?

Because Steve wanted to have the $30000 more than he wanted Valerie to be Steve's captive.

In this narrative, "Steve kidnapped Valerie" is interpreted as describing an instance of CaptureBargain in which Steve is the actor and Valerie is the victim. The assertion that "Mike paid Steve $30000" matches a part of the expansion of this instance of CaptureBargain, and the input "Mike is Valerie's father" implies the precondition that Mike values her freedom more than material possessions. If the definition of "kidnap" is removed from the lexicon, the narrative no longer directly references CaptureBargain nor even suggests it, and GENESIS can no longer understand the story as illustrated by the following trace:

Input: Steve kidnapped Valerie. Mike was Valerie's father and paid Steve $30000.

Thematic goal achieved: Steve is happy that Steve has the $30000.

Ready for questions:

>Summarize

Steve did something to Valerie. Mike gave Steve $30000.

>Why did Steve kidnap Valerie?

Cannot find sufficient reason.

>Why did Mike pay Steve the money?

Cannot find sufficient reason.

The same technique used to learn a definition for "kidnap" has also been used to acquire a preliminary definition for the schema label "entrap." The details of this example are presented in the discussion of the acquisition of a solicitation-for-entrapment schema presented in appendix B.

12.4. Comments on the Integrated Learning of Words and Concepts

Procedures in GENESIS for the integrated learning of words and their underlying concepts represent only a preliminary exploration of the potential interaction between language and concept acquisition. As such, there are many refinements that could be made and many problems that need to be addressed. Below is a short list of some of the obvious areas for future research.

(1) The procedure for removing schema suggestions from new definitions is too strict. One counter-example should not eliminate a suggestion and repeated usefulness of a suggestion should make it resistant to elimination.

(2) Morphology of unknown words should be considered. A "kidnapper" is clearly the actor of a "kidnapping."

(3) More and better heuristics are needed for determining whether a word might be a *schema label* and to what schema it might refer.

(4) Only *role labels* and *schema labels* are considered. Many words do not fall into either of these two categories.

(5) Only integration with explanation-based learning has been considered. Integration with similarity-based learning should also be examined.

CHAPTER 13

GENESIS: LEARNED SCHEMATA AS RETRIEVAL INDICES

Conceptual information retrieval involves indexing and retrieving textual information based on an interpretation of its "meaning." As discussed in [Schank81a], this approach has a number of advantages over standard information retrieval systems based on keywords. A conceptual information system like the CyFr system presented in [Schank81a] indexes specific textual passages under the schemata that an understanding system used in processing the text. These schemata can then act as indices for the retrieval of information by a question answering system. In CyFr, FRUMP [DeJong82b] was used to process news stories and CYRUS [Kolodner84] was used to index and retrieve episodes for the purpose of answering questions.

CYRUS was also capable of learning specializations of existing schemata and using these new schemata to index and retrieve specific events. For example, CYRUS started with a general schema for diplomatic meetings and then learned a specialization in which military aid was the topic of discussion. Specific episodes involving meetings about military aid were then indexed under this new schema, and this indexing was used to retrieve answers to questions such as: "Who has Vance talked to about military aid?" The ability to learn new ways of indexing episodes is important in maintaining an efficient and useful dynamic memory [Schank82]. It is crucial for maintaining organization in very large databases of information. Learning new indices also allows a system to notice similarities among texts unforeseen by the system's implementors and use these similarities to help retrieve related information [Mooney87b].

Like CyFr, GENESIS indexes specific episodes under schemata that were used in understanding the episode. In addition, since GENESIS can learn new schemata, the system is also capable of learning new ways of indexing episodes. However, GENESIS uses explanation-based methods to learn schemata for novel combinations of existing schemata, unlike CYRUS, which used empirical methods to learn specializations. A number of *case-based reasoning* systems also learn new schemata for indexing episodes in memory [Bain86, Kolodner87]. However, like CYRUS, these systems create schemata by noticing similarities between specific cases that were stored in the same place in memory. GENESIS, on the other hand, creates a schema from a single instance and actually uses this schema to notice the similarity between two instances and therefore store them in the same place in memory. Therefore, the learning of memory indices in GENESIS is very different from that performed by systems using case-based reasoning. The remainder of this chapter describes the simple indexing scheme used in GENESIS and gives examples of how it can be used to retrieve related episodes.[1]

When GENESIS has finished processing a narrative, the indexer examines the highest-level explanation for each of the thematic goals achieved in the narrative. It then stores the complete causal model created for this specific narrative in the long-term store, indexing it under each of the schemata in these highest-level explanations. An exception to this rule is if the system acquires a new schema from an explanation, in which case the episode is indexed under the new schema instead of under the schemata that compose it. When answering questions about a particular narrative, the system can be told to "Review similar stories." This causes the system to retrieve past episodes that are indexed under the same schemata used to index the present story and make them available for question answering and paraphrasing. Reviewing is done by temporarily replacing the causal model of the current text with the causal model previously constructed and saved for the episode being reviewed. For example, consider the state of the system after it has processed both the learning and test narratives for both the kidnap and murder examples. The following trace shows the behavior of the system when processing a third murder-for-inheritance story:

Input: Mary is the Bob's mother and is a millionaire. Someone strangled Mary. Bob got $1000000.

Thematic goal achieved: Bob is happy that Bob has the $1000000.

Ready for questions:

>Summarize
Bob murdered Mary. Mary had $1000000. Bob was Mary's heir. Bob inherited Mary's $1000000.

>How did Bob murder Mary?
Bob strangled Mary.

>Why is Bob Mary's heir?

[1] A description of GENESIS' ability to use learned schemata to index episodes is also presented in [Mooney87b].

Because Mary was Bob's mother.

>Review similar stories

There are 2 other instances of this schema. Enter number of story to be reviewed> 1

Reviewing MurderInheritStory1

Ready for questions:

>Summarize

Agrippina murdered Claudius. Claudius had an estate. Agrippina was Claudius's heir. Agrippina inherited Claudius's estate.

>How did Agrippina murder Claudius?

Agrippina poisoned Claudius with the mushroom.

>Why is Agrippina Claudius's heir?

Because Claudius was Agrippina's husband.

>

There are 2 other instances of this schema. Enter number of story to be reviewed> 2

Reviewing MurderInheritStory2

Ready for questions:

>Summarize

Gene murdered Martha. Martha had $1000000. Gene was Martha's heir. Gene inherited Martha's $1000000.

>How did Gene murder Martha?

Gene shot Martha and killed her.

>

There are 2 other instances of this schema. Enter number of story to be reviewed>

Review finished.

>Review kidnapping stories

There are 2 instances of CaptureBargain. Enter number of story to be reviewed> 1

Reviewing CaptureBargainStory1

Ready for questions:

>Summarize

Mary was free. John captured Mary and locked her in a room. John contacted Fred and told him that Mary was John's captive. John wanted to have $250000 more than he wanted Mary to be John's captive. Fred had a positive relationship with Mary. Fred had the $250000. John and Fred carried out a bargain in which John released Mary and Fred gave John the $250000 at Trenos.

>How did John communicate to Fred?

John called Fred and told him that Mary was John's captive.

>Why did Fred want Mary free?

Because Fred was Mary's father.

>

There are 2 instances of CaptureBargain. Enter number of story to be reviewed>

Review finished.

An additional feature that is illustrated in this trace is GENESIS' ability to use learned words in the retrieval of past episodes. For example, given the definition acquired for "kidnap" and some knowledge of English morphology, the system interprets: "Review kidnapping stories" as a command to retrieve past instances of the CaptureBargain schema.

CHAPTER 14

THE PSYCHOLOGICAL STATUS OF EXPLANATION-BASED LEARNING

Most theories of concept and schema acquisition in cognitive psychology have been similarity-based and postulated learning mechanisms based on inductive generalization across numerous examples (e.g. [Anderson79, Medin87b, Posner68, Rumelhart78, Thorndyke79]). However, several recent articles [Medin87a, Murphy85, Schank86b] have criticized this approach for its failure to acknowledge the importance of subjects' existing knowledge and its inability to explain subjects' preference for correlations that have causal explanations.

This raises the interesting question of whether an explanation-based learning mechanism can be productively interpreted as a cognitive model of certain types of human learning. Although a number of psychological experiments demonstrate people's ability to learn concepts or schemata from two examples using analogy [Gick83, Spencer86] or from many examples using similarity-based induction [Medin87b, Posner68], until very recently, there were no experiments that demonstrated people's ability to learn a concept or schema by explaining a single example. This chapter reviews some recent experiments that indicate that people, like GENESIS, can learn a plan schema by explaining and generalizing a single specific instance presented in a narrative. These experiments are more completely reviewed in [Ahn87a] and details on the experimental methodology are given in [Ahn87b].

14.1. Overview of the Experiments

The overall design of the experiments involved subjects reading a single narrative describing a specific instance of a novel plan and performing a variety of tests constructed to determine whether or not they had acquired an abstract schema from this single example. Three passages were constructed to present situations for which the subjects presumably did not already have a pre-established schema but which they could understand using plan-based understanding. For example, one passage involves a cooperative buying scheme used in other countries. In Korea the system is called a "Kyeah" and in India it is called a "chit fund". The experimental narrative describing a single instance of this plan follows:

Tom, Sue, Jane, and Joe were all friends and each wanted to make a large purchase as soon as possible. Tom wanted a VCR, Sue wanted a microwave, Joe wanted a car stereo, and Jane

> wanted a compact disk player. However, they each only had $50 left at the end of each month after paying their expenses. Tom, Sue, Jane, and Joe all got together to solve the problem. They made four slips of paper with the numbers 1,2,3, and 4 written on them. They put them in a hat and each drew out one slip. Jane got the slip with the 4 written on it, and said, "Oh darn, I have to wait to get my CD player." Joe got the slip with the 1 written on it and said, "Great, I can get my car stereo right away!" Sue got the number 2, and Tom got number 3. In January, they each contributed the $50 they had left. Joe took the whole $200 and bought a Pioneer car stereo at Service Merchandise. In February, they each contributed their $50 again. This time, Sue used the $200 to buy a Sharp 600 watt 1.5 cubic foot microwave at Service Merchandise. In March, all four again contributed $50. Tom took the money and bought a Sanyo Beta VCR with wired remote at Service Merchandise. In April, Jane got the $200 and bought a Technics CD player at Service Merchandise.

In addition to a group given specific narratives (the instance group), some experiments also used a control group that was given abstract descriptions of the schemata underlying each of the example narratives (abstract group). The description of the Kyeah schema given to the abstract group follows:

> Suppose there are a number of people (let the number be n) each of whom wants to make a large purchase but does not have enough cash on hand. They can cooperate to solve this problem by each donating an equal small amount of money to a common fund on a regular basis. (Let the amount donated by each member be m.) They meet at regular intervals to collect everyone's money. Each time money is collected, one member of the group is given all the money collected (n X m) and then with that money he or she can purchase what he or she wants. In order to be fair, the order in which people are given the money is determined randomly. The first person in the random ordering is therefore able to purchase their desired item immediately instead of having to wait until they save the needed amount of money. Although the last person does not get to buy their item early, this individual is no worse off than they would have been if they waited until they saved the money by themselves.

Since subjects in the abstract group had been directly told the content of the schema, they were presumed to have learned the schema. Consequently, if the instance group performed as well as the abstract group on a task requiring knowledge of the general schema, then it is reasonable to assume that the subjects in the instance group had also acquired the schema.

Before any of the experiments were conducted, the experimenters analyzed the instance passages from an EBL perspective and determined a set of variables and constraints characterizing the schema that could be learned from generalizing an explanation of the narrative. A *constraint* in this context is defined as a property that supports the explanation for how the thematic goals in the narrative were achieved. For example, the variables and constraints for the Kyeah passage are shown in Table 14.1. In order to determine whether a learned schema agreed with that predicted by EBL, subjects' learning in each task was judged based on how well they obeyed the constraints and recognized the mutability of the variables.

14.2. Experiment 1: Abstract Description Generation

The first experiment investigated whether subjects could acquire a schema from a single example by asking them to "write, in abstract terms, a description of the general technique illustrated in the narrative." Only the instance group was used in this experiment, since this task would simply be a memory test for the abstract group. In general, subjects in the instance group produced good descriptions of the schema. The following is the description of the Kyeah schema written by one subject:

> Suppose in a group of people, each person would like to buy something expensive, but over a period of time, each person cannot earn enough to buy what he would like. By using random selection, each person could be assigned a number. when the group had saved enough money *together* to purchase an item, the person with the first number would get his item. This would continue for the rest of the group until everyone had gotten what he wished.

Table 14.1: Variables and Constraints for the Kyeah Schema

Variables	Constraints
identity of participants	participants want items of similar value
number of participants (n)	participants cannot afford items
exact time of meetings	participants trust each other
interval between meetings (t)	participants can afford m each t
amount of donation (m)	each participant donates same amount
items bought	cost of desired items $\simeq$ n X m
stores where items bought	number of meetings = number of participants
method of determining order	order must be assigned randomly

A more objective measure of their performance is that, overall, subjects given a specific instance explicitly mentioned 75% of the constraints and identified 89% of the variables in their descriptions of the general schema. Details on the scoring methods used are given in [Ahn87b].

14.3. Experiment 2: New Instance Generation

The second experiment was constructed to determine whether subjects in both the instance and abstract group could produce another specific instance of the learned schema. Subjects in the instance group were told to "write another story in which characters use the general method illustrated in the story but that is otherwise as different as possible" while subjects in the abstract group were told to "write a story in which particular individuals use the technique described in the passage in a specific case." In general, both groups produced equally good narratives. The following is the new Kyeah narrative written by one subject in the instance group.

> Bill, Kim, John and Mary were all business associates. Bill wanted some land in Northern Illinois, Kim wanted a new house in Switzerland, John wanted a new Porshe 928S with all all accesories, and Mary wanted to take a trip around the world. The only problem was they each only had $25,000.00 left unspent at the end of each month. They all got together and picked random variables on Bill's business computer. Mary was farthest from her variable so she would have to wait till last to get her trip around the world. John nailed his variable and jumped enthusiastically saying, "Yea, I get to get my new Porshe 928S right now." They each talked with their banker and drew the $25 Thousand dollars out and pooled it together after the first month and the next day John drove up in his new, black, 928S with all accessories. At the end of the next month they again pooled their money and Kim got her chalet in Switzerland. Again at the end of the next month they pooled their money an Bill got his land in Northern Illinois. Finally, after the fourth month they pooled their money together and Mary left for her trip around the world.

A more objective measure of their relative performance is that the stories written by the instance group explicitly and correctly instantiated 81% of the constraints compared to 76% for the abstract group. The difference between the two groups is insignificant, indicating that subjects in both groups learned the schema equally well.

14.4. Experiment 3: Yes/No Questions

The third and final experiment directly tested whether subjects could correctly identify whether or not a particular component was a variable or a constraint in the general schema. Subjects in both the instance and abstract group were explicitly asked yes/no questions like: "Can some people consistently donate less than others and have the system work?" and "Is there any particular number of people required for this plan?" Overall, the two groups were able to answer these questions equally well. The instance group correctly answered 85% of the questions compared to 81% for the abstract group.

14.5. Discussion of the Experimental Results

In general, the experiments reviewed in this chapter support the claim that people can learn a schema by explaining and generalizing a single narrative à la GENESIS. However, also like GENESIS, people can only perform explanation-based learning when they can explain all of the actions that compose the schema. In [Ahn87b], additional experiments are presented that demonstrate subjects' inability to learn schemata from narratives for which they cannot construct complete causal explanations. For example, subjects could not learn a schema for an American Indian potlatch ceremony from one example since they could not determine motivations for all of the actions in the example.

There are also some interesting observations that can be made regarding subjects' apparent generalization processes. For example, unlike SOAR (see section 3.8), they obviously do not simply change each constant in the explanation to an independent variable. Since in the Kyeah narrative all of the characters made their purchases at the same store (i.e. Service Merchandise), simply changing the constant representing this store to a variable would result in a schema in which all of the participants had to buy their items at the same store. However, in none of the experiments did any of the subjects believe that this was a constraint on the underlying schema. Of course, this does not imply that subjects must therefore use a generalization process analogous to the unification algorithms discussed in chapter 3; however, it does indicate that people's generalization processes are not as simple as changing constants to variables. Subjects' generalization process takes greater advantage of their existing knowledge of the domain. In fact, it is clear that in some ways people's ability to generalize explanations is even more powerful than algorithms like EGGS, EBG, and STRIPS. For example, in the Kyeah example, almost all of the subjects realized that the plan would work with any number of people as long as there was an equivalent number of meetings for collecting and allocating the money. However, producing this generalization requires recognizing the repeated structure of the original example and producing a general iterative plan for the overall schema. Performing this process of *generalizing to n* from one example in a machine learning system requires additional generalization techniques. BAGGER [Shavlik87a, Shavlik88] is an EBL system which, in addition to EGGS, uses a process for analyzing recursive rule applications in order to learn iterative plans from one example.

CHAPTER 15

CONCLUSIONS AND FUTURE WORK

This thesis has demonstrated that a general explanation-based learning mechanism can efficiently learn and improve its performance in a wide variety of domains. In particular, it has shown that a general EBL mechanism is capable of improving performance on the complex task of understanding narrative text by acquiring schemata for novel plans that achieve important goals. Finally, it has reviewed empirical evidence that indicates that explanation-based learning is a plausible model of certain types of human learning in this domain. This final chapter summarizes the unique features of the research presented in this thesis and outlines some problems requiring future research.

15.1. Relation to Other Work

Below are four aspects that make the work presented in this thesis unique:

(1) It is based on a general domain-independent explanation-based learning mechanism which has been tested on numerous examples from different domains.

(2) It has addressed the problem of generalizing the temporal order of actions in plan-based explanations.

(3) It has illustrated the ability of EBL to improve the performance of an understanding system as well as a problem-solving system.

(4) It has examined the psychological plausibility of an explanation-based learning system.

15.1.1. Generality and Domain-Independence

Earlier explanation-based learning systems were constrained by the particular domains to which they were applied and were never tested on examples from a variety of domains. This includes such systems as STRIPS [Fikes72], LEX2 [Mitchell83], CUPS [Winston83], LP [Silver83], MA [O'Rorke84], ARMS [Segre85], LEAP [Mitchell85], PHYSICS-101 [Shavlik85], and the original GENESIS [Mooney85a].

More recent EBL systems such as PROLOG-EBG [Kedar-Cabelli87a], MRS-EBG [Hirsh87], PROLEARN [Prieditis87], and PRODIGY [Minton87a, Minton87b] are general domain-independent EBL systems; however, these systems have not been tested on as wide a variety of examples.

According to [Kedar-Cabelli87a], "PROLOG-EBG has been tested on the 'cup' and 'safe-to-stack' examples, and the 'suicide' example" (p. 389). In [Hirsh87], the examples presented are the Safe-To-Stack and LEX2 examples; no other examples are mentioned. In [Prieditis87], the Safe-To-Stack and LEAP examples and examples on list membership and Towers of Hanoi are the only ones mentioned. In [Minton87b], it states that PRODIGY has been tested on machine shop scheduling and 3-D robotics construction as well as the "blocks world."

In addition, none of these systems support all of the representational formalisms supported in EGGS (i.e. Horn clauses, rewrite rules, and STRIPS operators). PROLOG-EBG, MRS-EBG, and PROLEARN only use Horn clauses and PRODIGY supports STRIPS operators and Horn clauses. Also, except for PRODIGY, none of these systems have been integrated with a performance system operating in a complex domain. Finally, unlike EGGS, none of these systems have facilities for learning from the observed problem solving behavior of other agents.

15.1.2. Generalizing Temporal Ordering

Previous research on EBL and macro-operators has not addressed the problem of learning macro-operators with partially ordered actions. STRIPS [Fikes72] and most other systems that learn macro-operators (e.g. [Iba85, Korf85, Minton85]) learn linear sequences. Some recent work has addressed the issue of learning iterative macro-operators [Cheng86, Prieditis86, Shavlik87a]. However, the packaging procedure presented in section 6.2.2 is apparently the first system that learns macro-operators with partially-ordered actions. Although the procedure for generating partially-ordered macro-operators is closely related to certain processes in nonlinear planning, there are a number of important differences which were discussed in section 6.2.2.6.

15.1.3. Learning for Understanding

Unlike GENESIS, almost all machine learning systems with performance components use learning to improve the abilities of a problem-solver or planner rather than an understander. IPP [Lebowitz80, Schank82] is still one of the few machine learning systems that improved its ability to understand. IPP processed news articles and used a similarity-based approach to learn specializations of existing schemata. For example, the system already had a schema for kidnapping and after processing several stories that described kidnappings in Italy carried out by the terrorist group the Red Brigades, it created a specialized schema for kidnappings in Italy in which the Red Brigades was the default kidnapper. Later, if it encountered an article describing a kidnapping in Italy in which the kidnappers were not mentioned, it assumed the Red Brigades was the responsible party. Besides using a similarity-based instead of an explanation-based approach to learning, IPP learned specializations of existing schemata instead of new schemata for compositions of actions that achieve important goals. As mentioned in chapter 13, CYRUS [Kolodner84] also used similarity-based methods to learn specializations of existing schemata, which it used to index and retrieve specific episodes.

Other than GENESIS, the only EBL system that improves its ability to understand external problem solving behavior is the ARMS system [Segre87a]. ARMS uses learned schemata to aid the understanding of assembly sequences in a robotics domain. However, as mentioned in section 10.1.3, the understanding system in ARMS is incapable of inferring missing actions and therefore, unlike GENESIS, the plan schemata it learns can not be used to fill in gaps in future observations.

15.1.4. Psychological Plausibility

This research is also unique in that it has motivated psychological experiments that demonstrate that people can perform explanation-based learning from a novel plan presented in a narrative [Ahn87a, Ahn87b]. There are currently no other psychological experiments specifically directed at judging the ability of an EBL system to model human learning. Nevertheless, as reviewed in [Murphy85], there is a substantial amount of psychological research that reveals the important effect subjects' background knowledge and naive theories of the world have on the process of concept acquisition.

15.2. Problems for Future Research

The research presented in this thesis as well as numerous related projects has revealed a number of interesting problems that need to be addressed. This section briefly discusses several problem areas in explanation-based learning and mentions current research efforts that are attempting to confront these problems.

15.2.1. The Effect of Learning on Performance

As illustrated with empirical data in chapter 7, explanation-based learning can have both positive and negative effects on future performance. However, for unrestricted learning of macros in problem solving, there is conflicting empirical evidence regarding which of these effects dominates in the long run [Minton85, O'Rorke87a, Shavlik88]. Consequently, further empirical and theoretical analysis of the effect of learning on performance is required. Of course, learning and forgetting the right macros is crucial to performance improvement.

15.2.2. Operationality and Pruning

Learning macros or schemata at the appropriate level of generality is one aspect of insuring that learning improves performance. However, there is currently no general domain-independent characterization of what makes a concept operational. In EGGS, pruning for operationality requires a special procedure for each domain. In PROLOG-EBG, arbitrary axioms are used to define particular predicates as operational and in MRS-EBG, an arbitrary set of axioms can be used to prove that an expression is operational. While these approaches allow for flexibility in determining operationality, they do not help to characterize the notion of operationality in a general way. Some recent research in EBL has been directed at the problem of determining operational descriptions. Systems in particular domains have been used to explore learning schemata at different levels of generality [Segre87b, Shavlik87b] and the METALEX system [Keller87a] searches for specific concept descriptions that empirically improve performance. However, there is currently no analytical characterization of operationality. In fact, current experience probably indicates that such a characterization is not forthcoming and that heuristic and empirical methods will continue to be the only ways of determining an appropriate level of generality for what is learned.

15.2.3. Deciding What to Learn

A slightly different problem is determining whether or not a particular macro or schema is worth learning at all. Learning large numbers of useless rules only exacerbates the problem of slowdown. As mentioned in chapter 7, there has been some research on analytical and empirical methods for determining utility. In particular, this has beem the primary focus of Minton's research [Minton85, Minton87a]. However, the problem of determining what must be learned and what must be forgotten in order to improve overall system performance is still largely an open question.

15.2.4. Understanding Observed Behavior

Learning from observed problem solving behavior like that performed by GENESIS and various *learning apprentice* systems [Mitchell85, O'Rorke87b, Segre87a, Wilkins86], is an efficient method for acquiring expert knowledge. Since this process requires the ability to explain the intentional actions of external agents, the process of understanding or *plan recognition* [Schmidt78], becomes an important part of the learning process. However, understanding is a difficult process which is far from completely understood despite the fact that it has been fairly extensively studied in natural language processing. The understanding component in GENESIS and other current systems is clearly incapable of explaining the wide variety of situations that humans can comprehend. Of course, research in this area is continuing and there has recently been some new approaches to the problem. Recent work by Charniak [Charniak86] and Kautz and Allen [Kautz86] were briefly discussed in section 10.1.3. Recent understanding research by Schank and his colleagues [Kass86, Leake86, Schank86a] has involved the SWALE system, which constructs explanations for situations by modifying or *tweaking* existing schemata or *explanation patterns*. This approach is an attempt to avoid the combinatorially explosive nature of plan-based understanding while maintaining the ability to explain novel situations.

15.2.5. Intractable Domain Theories

In EBL, the combinatorially explosive nature of explaining why a plan works or why an example is a member of a concept has been referred to as the *intractable theory problem* [Mitchell86]. In any realistic domain, constructing a complete explanation for why a plan actually works is an expensive process. For example, the kidnapping schema GENESIS learns is actually overly-general because it does not consider the possibility that the police or other counter-agents might interfere with the plan. The act of communication between the kidnapper and the ransom payer should actually be constrained so that it does not reveal the kidnapper's identity and thereby result in his arrest. GENESIS is implicitly making the assumption that external agents will not interfere; however, it has no way of recovering if this assumption is violated. Attempting to constrain the plan to prevent any possible interference on the part of any other agent is clearly not a tractable solution. Systems that attempt to deal with this problem are described in [Chien87a, Chien87b]. The basic approach taken in Chien's work is to make simplifying assumptions during the explanation process in order to make it tractable. When a subsequent failure occurs, it is explained in terms of a violation of an initial assumption and the learned concept is refined to account for the failure. Alternative approaches to dealing with the intractable domain theory problem are described in [Bennett87, Doyle86, Tadepalli86].

15.2.6. Incomplete Domain Theories

Corresponding to the intractable theory problem is the *incomplete theory problem* [Mitchell86] in which the current domain theory is incapable of completely explaining an example. One approach to refining an initial domain theory based on experimentation is described in [Rajamoney85, Rajamoney87]. When a prediction supported by the current theory is contradicted by empirical evidence, the system examines the domain axioms underlying the faulty prediction and conducts experiments to determine exactly which of these axioms is incorrect. Another approach to dealing with incompleteness is to use similarity-based methods to induce initial causal theories [Anderson87a, Pazzani87].

15.2.7. Hybrid Learning Methods

Approaches to dealing with intractable and incomplete theories frequently involve integrating explanation-based and similarity-based learning.[1] For example, integrated approaches can be used to deal with intractability by using the detection of similarities to focus the explanation process. This approach is taken in [Lebowitz86], in which similarities in the congressional voting record are used to focus the process of explaining why certain members of congress voted for or against a particular bill. If the incompleteness arises from the fact that operators are represented procedurally instead of declaratively, empirical techniques can be used in the actual process of generalizing explanations [Porter85, Porter86].

Correspondingly, EBL methods can be used to address problems with purely empirical approaches. For example, explanations of particular examples can be used to select relevant features for similarity-based methods [Danyluk87, Flann86, Mitchell84]. Also, goal-related knowledge can be used to guide feature selection for conceptual clustering systems [Mogensen87, Stepp86]. In general, this approach allows explanations to contribute to a similarity-based system's *inductive bias* [Utgoff86].

Current approaches to combining the two learning methods effectively use one of the methods to focus or direct the other. An interesting area for future research involves using each method to learn different parts or features of a single concept. For example, imagine a system like GENESIS trying to learn a schema for a birthday party by reading narratives about particular celebrations. Such a system could probably use its domain knowledge to explain the baking, cutting, and eating of the birthday cake since these actions are causally connected and satisfy important hunger and enjoyment goals. However, it would probably not be able to explain why someone put a particular number of candles on the cake and why someone else made a wish and then blew them out while everyone else sang. However, these components of the birthday-party schema might be learned using a similarity-based approach. Since many stereotypical actions (e.g. a wedding ceremony or a trip to the restaurant or supermarket) contain both features that are causally necessary and others

[1] Some researchers may suggest that EGGS would combine nicely with BACON [Langley81]; however, four out of five doctors would probably agree that this is not a healthy combination.

that are conventional, such an approach to integrating the two methods could be very useful, particularly in the domain of narrative understanding.

15.3. Conclusions

This thesis has shown that a single general learning mechanism can perform explanation-based learning in a wide variety of domains. The generalization mechanism in the EGGS system has been shown to be able to generalize explanations based on logical proofs, term rewritings, and plans composed of STRIPS operators. This generalization algorithm was compared to the generalization algorithms in STRIPS and PROLOG-EBG. Like EGGS, both of these algorithms use unification pattern matching to compute the most general explanation that maintains the structure of an explanation for a specific example. Theoretically, the time required to compute a generalized explanation was shown to be linear in the size of the explanation. Compared to generalization algorithms based on unification, the generalization algorithm in SOAR was shown to be susceptible to under-generalization.

With regard to logical proof explanations, mechanisms in EGGS for proof construction, proof completion, and macro-rule learning were presented. Examples of learning using logical proof explanations were given for the following domains: logic circuit design (as in the LEAP system), logic theorem proving (as in MA), integration problem solving (as in LEX2), geometry theorem proving (as in ACT* and PUPS), and blocks world planning (as in STRIPS).

Regarding explanations composed of rewrite rules, mechanisms in EGGS for rewriting expressions and learning rewrite macro-rules were discussed. Examples of learning using term rewritings were presented for the following domains: logic circuit design, integration problem solving, and equation solving.

Finally, regarding plan-based explanations composed of STRIPS operators, mechanisms in EGGS for verifying plans and learning macro-operators with partially ordered actions were presented. The process of learning partially ordered macro-operators from specific plans was compared to the task of nonlinear planning. Examples of learning using plan-based explanations were given for blocks world planning and computer programming.

Empirical results on the effect of macro-rules on problem solving performance were also presented. These results show that learning macro-rules can greatly improve performance on similar problems; however, in certain situations, it can also substantially degrade performance. Approaches to dealing with the problem of degraded performance were reviewed and suggestions for conducting properly controlled experimental comparisons of learning and non-learning systems were given.

The EGGS learning system has been tested most thoroughly as a component of the GENESIS narrative understanding system. The overall architecture of GENESIS was reviewed and the importance of learning to improve the performance of a complete AI system was discussed. The knowledge representation used in GENESIS and its ability to construct explanations for narratives were discussed in detail using the learning of a murder-for-inheritance schema as an example. The schema and plan-based understanding processes in GENESIS were presented and compared to those in previous narrative understanding systems.

It was shown how schema acquisition in GENESIS uses the EGGS system to generalize the explanations constructed for characters' actions in a narrative. An additional generalization step that insures that the learned schema represents a volitional action and does not contain any unmotivated actions was also discussed. The description of schema acquisition in GENESIS also included discussions of processes for deciding when to learn, packaging a generalized explanation into a schema, and indexing a schema so that it can be used to aid subsequent understanding.

In addition, GENESIS' ability to learn provisional meanings for unknown schema-related words was reviewed. Unlike other approaches to the acquisition of word meaning, GENESIS is capable of simultaneously learning a concept and meanings for related words from a single example. GENESIS's ability to use learned schemata to index and retrieve specific episodes was also presented.

Finally, psychological experiments were reviewed that revealed that human subjects, like GENESIS, could learn a novel plan schema from a single example. These experiments demonstrated that after reading a specific instance of a novel plan, subjects could write a good description of the underlying schema. They could also produce another instance of the schema and answer questions about the schema as well as subjects who read a passage directly describing the abstract plan.

Together with other recent projects investigating general EBL mechanisms, the work presented in this thesis is an indication that research in EBL has reached a certain level of "maturity" and has moved beyond the stage of exploratory programs that operate in a single domain. Nevertheless, as outlined in the previous section, there continue to be even more interesting issues on the horizon which are already attracting a great deal of attention.

APPENDIX A

LINEAR SUBSTITUTION APPLICATION

Although a linear-time unification algorithm is described in [Paterson78], a linear-time algorithm for applying a substitution to a wff is required in order insure that a generalized explanation can be computed in linear time (see section 3.4). To insure linear performance, a substitution application algorithm must be careful to avoid tracing down the final value of a variable multiple times. Such duplication of work can be prevented by using *path compression* like that performed in the standard UNION-FIND algorithm [Reingold77]. One approach is to first compress or "flatten" a substitution so that every variable in the substitution is directly bound to its final value. Since a flattened substitution can easily be applied to a wff in linear time, if the flattening can be performed in linear time, then the complete process will be linear.

As in [Paterson78], it is assumed that wffs are represented as directed acyclical graphs in which common subexpressions are represented by a single subgraph. A node representing a k-ary function or predicate has outdegree k and $son(n, i)$ $(1 \leqslant i \leqslant k)$ refers to the ith son (argument) of node n. A substitution will consist of a list of nodes representing variables that have pointers to nodes representing their values (i.e. $Value(v)$ refers to the value of variable node v and is NIL if v is not bound). It is assumed that this substitution is generated by a unifier and therefore does not have any occur-check violations. Given this representation, an algorithm for applying a substitution to a wff is given in Figure A.1.

The procedure FlattenSubstitution changes the substitution into one in which variables point directly to their final values. This is done by first finding the final value for each variable by following value pointers and performing *path compression* [Reingold77] so that all variables encountered along the way are also made to point to their final values. Finding the final value of any variable will be called a FIND since it is analogous to the corresponding process in the UNION-FIND problem. Performing a FIND on all the variables in the substitution is linear in the size of the substitution for the following reasons. Let n be the number of variables in the substitution (the total number of value pointers) and let p be the total number of value pointers followed by doing a FIND on all variables. Every FIND requires following exactly one last pointer to a value that is either an unbound variable or not a variable at all (let such pointers be called *final pointers*). Consequently, following final pointers accounts for n of the total number of pointer references. Let o be the total number of pointers traversed that are *not* final pointers. Therefore, $p = n + o$.

```
let S be the substitution and w be the wff to which it is to be applied
FlattenSubstitution(S)
ReplaceVariables(w)

procedure FlattenSubstitution(S)
   for variable v in S do
      let A be an empty stack
      while v is a variable node and Value(v)≠NIL do
         push(v, A)
         let v = Value(v)
      while A is nonempty do
         Value(pop(A)) = v
   for variable v in S do
      if Value(v) is not marked as already visited then
         mark Value(v) as visited
         ReplaceVariables(Value(v))

procedure ReplaceVariables(n)
   if n is a node for a variable or a constant
      then return
      else
         for i from 1 to outdegree(n) do
            if son(n,i) is a variable node and Value(son(n,i))≠NIL
               then let son(n,i) = Value(son(n,i))
               else ReplaceVariables(son(n,i))
```

Figure A.1: Linear Substitution Application Algorithm

Each of the o non-final pointers followed in a FIND results in that pointer being deleted and being replaced by a final pointer. Since the initial number of non-final pointers is clearly less than n (the total number of pointers), there are at most n non-final pointers to be deleted and hence $o \leqslant n$. Consequently, $p \leqslant 2n$ and therefore finding the final values for all the variables takes only linear time.

The procedure ReplaceVariables simply traverses a wff and replaces any bound variables encountered with their previously computed final value. Since traversing a graph is linear in the size of the graph, this process is clearly linear in the size of the input wff. After the final values for all variables has been computed, ReplaceVariables is called on all the final values in order to replace any variables within them with *their* final values. Values are marked when they are traversed to avoid traversing them more then once. Since ReplaceVariables is linear, the process of replacing variables in values is also linear with respect to the size of the overall substitution. This finishes the process of "flattening" and all variables are now bound to their completely instantiated final values. Calling ReplaceVariables on the input wff will, in linear time, replace all variables with their final instantiated values. Since "flattening" is linear in the size of the substitution and since replacing variables in the input wff is linear in the size of the wff, the algorithm in Figure A.2 is linear in the size of its inputs.

APPENDIX B

ADDITIONAL GENESIS EXAMPLES

In chapters 9-11 a detailed description was given of how GENESIS learns, indexes, and subsequently uses a murder-for-inheritance schema. In addition, in chapter 2, a trace of GENESIS' I/O behavior was given for the learning of a kidnap-for-ransom schema. This appendix provides further details on the kidnapping example as well as detailed descriptions of two additional GENESIS examples. The two additional examples include learning a schema for a person burning their own property in order to collect the insurance money and learning a schema for a police-officer impersonating a prostitute in order to arrest potential customers. Definitions for all of the schemata and rules needed to process these examples are given in appendix C.

B.1. The Kidnap Example

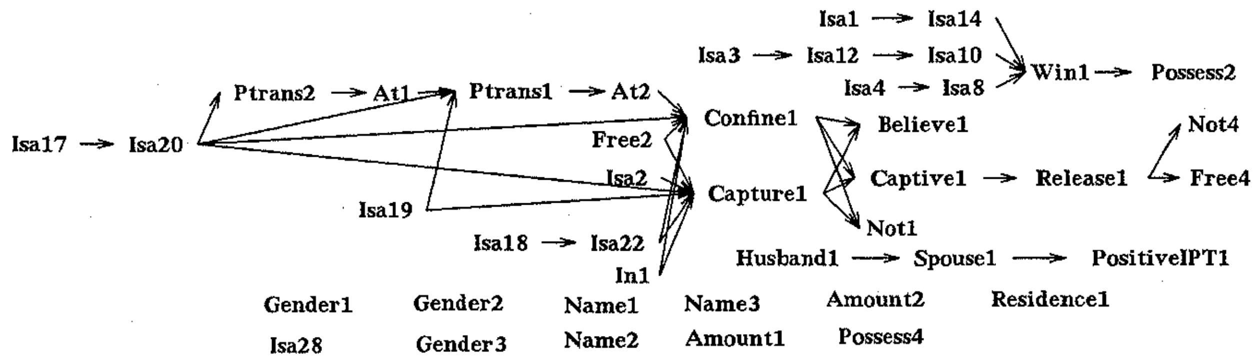
This section presents additional information on the kidnapping example. The test and learning narratives as well as a system trace for this example were presented in chapter 2.

B.1.1. Processing the Test Narrative Before Learning

The parsed version of the test narrative for the kidnap example is shown in Figure B.1. When processing this narrative for the first time given only the knowledge in appendix C, GENESIS constructs the causal model shown in Figure B.2. Except for connecting Alice's imprisonment to her subsequent release, relatively little causal structure is built. The fact that Bob locked Alice in his basement (which is parsed as a Confine action) suggests a Capture schema and this schema is

```
(Isa Person1 Person) (Gender Person1 male) (Name Person1 Ted) (Isa Person2 Person)
(Name Person2 Alice) (Gender Person2 female) (Husband Person1 Person2)
(Isa Money1 Money) (Amount Money1 100000 dollar) (Isa Lottery1 Lottery)
(Win Person1 Money1 Lottery1) (Isa Person3 Person) (Gender Person3 male)
(Name Person3 Bob) (Isa Basement1 Basement) (Isa Location1 Location)
(In Location1 Basement1) (Confine Person3 Person2 Location1 Basement1)
(Residence Person3 Basement1) (Isa Money2 Money) (Amount Money2 75000 dollar)
(Possess Person3 Money2) (Release Person3 Person2 ?FROM6 ?IN12)
```

Figure B.1: Parsed Version of the Kidnap Test Narrative



Free4	Alice is free.	Win1	Ted won the $100000 in the lottery.
Release1	Bob released Alice.	Isa8	The lottery is a game.
Captive1	Alice was Bob's captive.	Isa4	The lottery is a lottery.
Capture1	Bob captured Alice.	Isa10	The $100000 is an inanimate object.
In1	In the basement is in the basement.	Isa12	The $100000 is a valuable.
Isa22	The basement is a room.	Isa3	The $100000 is money.
Isa18	The basement is a basement.	Isa14	Ted is a character.
Isa19	In the basement is a location.	Isa1	Ted is a person.
Isa2	Alice is a person.	PositiveIPT1	Ted has a positive relationship with Alice.
Isa20	Bob is a character.	Spouse1	Ted is Alice's spouse.
Isa17	Bob is a person.	Husband1	Ted is Alice's husband.
Free2	Alice was free.	Isa28	The $75000 is money.
Not1	Alice was not free.	Gender1	Ted is male.
Confine1	Bob confined Alice in the basement.	Gender2	Alice is female.
At2	Alice is in the basement.	Gender3	Bob is male.
Ptrans1	Bob moved Alice into the basement.	Name1	Ted's name is Ted.
At1	Alice is at some place.	Name2	Alice's name is Alice.
Ptrans2	Bob moved Alice.	Name3	Bob's name is Bob.
Possess4	Bob has the $75000.	Believe1	Bob believes that Alice is Bob's captive.
Not4	Alice is not Bob's captive.	Amount1	The $100000 is 100000 dollars of money.
Possess2	Ted has the $100000.	Amount2	The $75000 is 75000 dollars of money.
		Residence1	Bob lives in the basement.

Figure B.2: Causal Model for Kidnap Test Example Before Learning

immediately determined since the Confine step is causally supported by all of the other steps in Capture. Bob's release of Alice is connected to his previous Capture since the Release's precondition of having Alice held captive is achieved by the Confine action in the Capture. As shown in Figure B.2, the fact that Bob has $75,000 is not connected to the rest of the actions in the narrative.

Due to this impoverished representation of the narrative, the system is unable to answer the questions presented in chapter 2. This example originally revealed an interesting problem with an early version of GENESIS' question-answering system. When asked: "Why did Bob lock Alice in his basement," it produced the reply: "So Bob could release Alice." It judged this to be a reasonable answer because one of its heuristics for answerering a *why* questions about an action was to find a later action that it enabled. This heuristic was later refined to eliminate enabled actions that simply re-achieve states that were already true before the execution of the action in question. This eliminates the answer: "So Bob could release Alice" since this release only achieves the state of having Alice free, which was already true before he locked her in the basement.

B.1.2. Processing the Learning Narrative

The parsed version of the learning narrative for the kidnap example is shown in Figure B.3 While processing this narrative, GENESIS builds a very large and interconnected causal structure. Pieces of this causal structure are shown in Figures B.4 - B.8. Figure B.4 shows the explanation for why Mary got in the car. This explanation can be summarized as follows. John's action of

---

```
(Isa Person7 Person) (Gender Person7 male) (Name Person7 Fred)
(Isa Person8 Person) (Name Person8 Mary) (Gender Person8 female)
(Isa Money3 Money) (Amount Money3 order-millions dollar)
(Father Person7 Person8) (Possess Person7 Money3) (Isa Person9 Person)
(Gender Person9 male) (Name Person9 John) (Isa Location2 Location)
(At Person8 Location2) (Ptrans Person9 Person9 ?FROM11 Location2)
(Isa Gun1 Gun) (Aim Person9 Gun1 Person8) (Color Jeans1 blue) (Isa Jeans1 Jeans)
(Attire Person8 Jeans1) (Isa Car1 Car) (Isa Location3 Location) (In Location3 Car1)
(Mtrans Person9 (Implies (Not (Ptrans Person8 Person8 ?FROM61 Location3))
                    (Shoot Person9 Person8 ?INSTRUMENT9)) Person8)
(Possess Person9 Car1) (Isa Hotel1 Hotel) (Isa Location4 Location)
(At Hotel1 Location4) (Drive Person9 Person8 ?FROM116 Location4 ?VEHICLE10)
(Residence Person9 Hotel1) (Isa Room1 Room) (Isa Location5 Location)
(In Location5 Room1) (Confine Person9 Person8 Location5 Room1)
(Residence Person9 Room1) (DialTelephone Person9 Person7 ?NUMBER36)
(Mtrans Person9 (Captive Person8 Person9 ?LOC15 ?IN18) Person7) (Isa Money4 Money)
(Amount Money4 250000 dollar) (Isa Restaurant1 Restaurant)
(Name Restaurant1 Trenos) (Isa Location6 Location) (At Restaurant1 Location6)
(Mtrans Person9 (Implies (Atrans Person7 Money4 Person9 Location6)
                    (Release Person9 Person8 ?FROM500 ?IN92)) Person7)
(Atrans Person7 ?PhysicalObject1 Person9 ?AT24)
(Release ?PhysicalObject2 Person8 ?FROM633 ?IN169) (Isa Person22 Person)
(Gender Person22 female) (Name Person22 Valerie) (Husband Person7 Person22)
(Mtrans Person7 (?Action1 ?Person1 Person8) Person22)
```

Figure B.3: Parsed Version of the Kidnap Learning Narrative

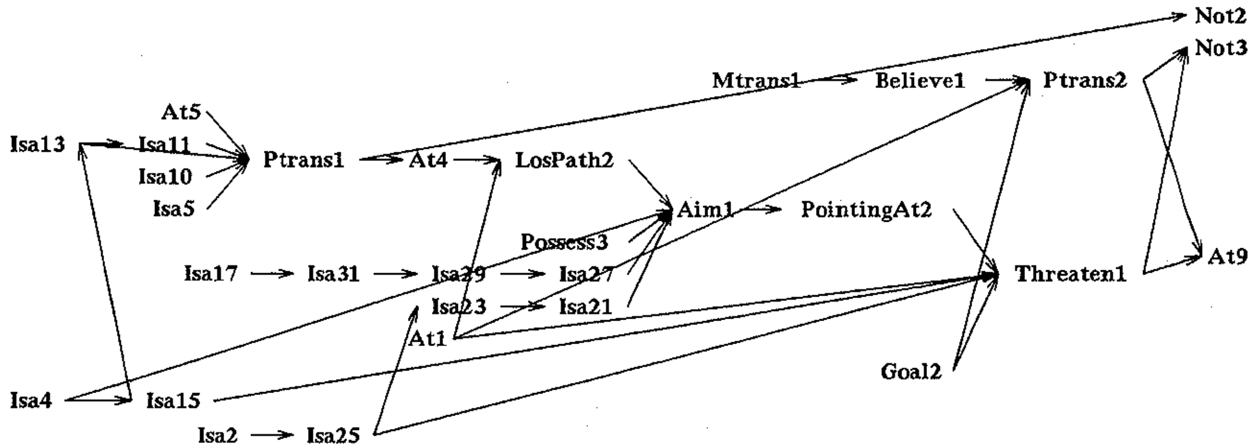
---

approaching Mary puts them both in the same location and enables him to aim his gun at her. John telling Mary that he would shoot her if she didn't get in the car suggests a Threaten schema. Since John driving Mary to the hotel requires as a precondition that Mary be in the car and since this state is the effect of the successful completion of the Threaten, the Threaten schema is determined. This in turn adds the assertion that Mary got in the car because she didn't want to be shot and causes the effect of the Aim to be equated to a precondition of the Threaten.

The remaining explanation for how Mary became John's captive (an expansion of the Capture schema) is shown in Figure B.5, which can be summarized as follows. John driving Mary to the hotel brings them all to the hotel's location. John confining Mary in the hotel room suggests a complete Capture schema, which is immediately determined since the Confine action is supported by all the other actions in Capture. This determination adds the assertion that John must have moved Mary into the room, which in turn is connected to the fact that she was already at the hotel.

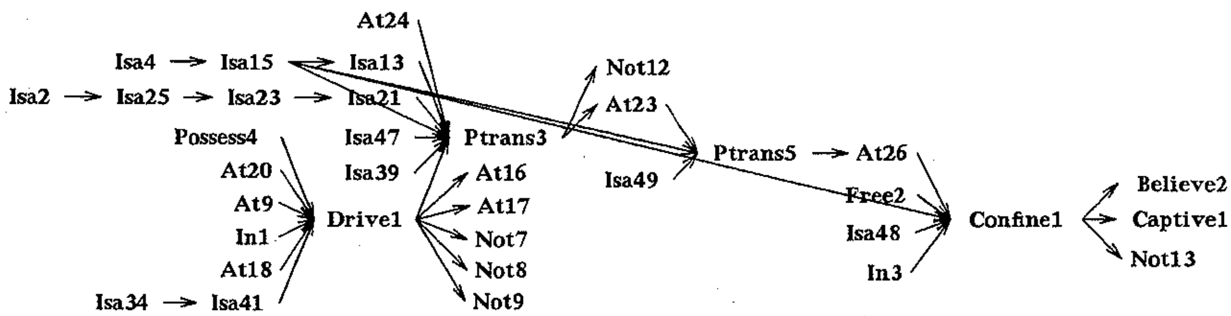
The complete highest-level explanation for how John got the $100,000 is shown in Figure B.6 and can be summarized as follows. John calling Fred suggests a Telephone schema, which is subsequently determined when John tells Fred he is holding Mary captive. This Telephone instance is possible because John believes that he is holding Mary captive as an effect of the Capture. It should be noted that GENESIS has a very naive view of communication in which believing a proposition is a precondition for communicating it and in which everyone believes everything that they are told. Next, John telling Fred that he would release Mary if Fred gave him $100,000 suggests a Bargain schema, which is determined when both of these actions take place. Preconditions of the completed Bargain include that John have Mary held captive (which matches an effect of the Capture), that Fred have the $100,00 (which is inferred from the fact that he is a millionaire), that Fred believe that John has Mary (which matches an effect of the Telephone), and that Fred value Mary's freedom more than $100,000 (which is inferred from the fact that he is her father). Figure B.7 shows the expansion of the Bargain schema and Figure B.8 shows all the facts in the narrative that remain unconnected to the rest of the text. The causal model constructed for this narrative and illustrated in Figures B.4 - B.8 allows GENESIS to answer the wide variety of questions presented in the trace in chapter 2.

At this point it is appropriate to note that the Bargain and Threaten schemata are more complicated to define than the other schemata in GENESIS. This is because the effects of these schemata depend on the actions that are involved. Consequently, a special construct (*Pointer*) is used in the definitions of these schemata in order to indirectly reference the effects and preconditions of



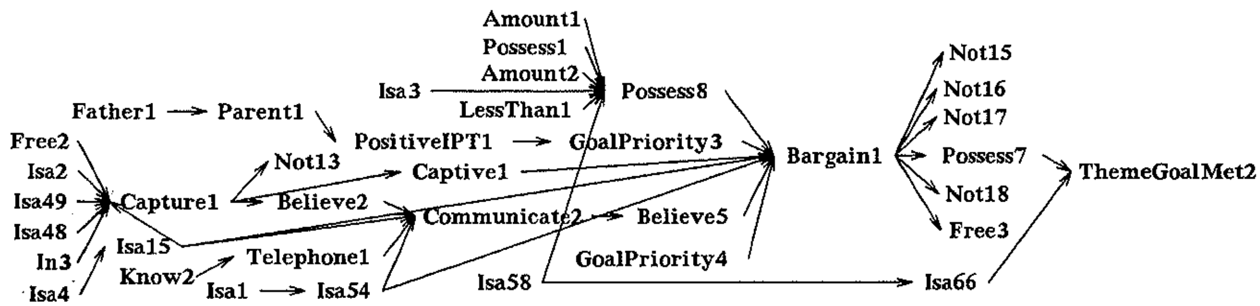
At9	Mary was in the car.
Threaten1	John threatened to shoot Mary with the gun unless Mary went from John into the car.
Isa25	Mary is a character.
Isa2	Mary is a person.
Isa15	John is a character.
Isa4	John is a person.
Goal2	Mary wants not to be shot.
At1	Mary was at John.
PointingAt2	John is pointing the gun at Mary.
Aim1	John aimed the gun at Mary.
Isa21	Mary is a physical object.
Isa23	Mary is an animate object.
Isa27	The gun is a physical object.
Isa29	The gun is an inanimate object.
Isa31	The gun is a weapon.
Isa17	The gun is a gun.
Possess3	John has the gun.
LosPath2	John has a line of sight path to Mary.
At4	John is at Mary.
Ptrans1	John went to Mary.
Isa5	Mary is a location.
Isa10	John is a location.
Isa11	John is a physical object.
Isa13	John is an animate object.
At5	John was at some place.
Not2	John is not at some place.
Not3	Mary is not at John.
Ptrans2	Mary went from John into the car.
Believe1	Mary believes that if Mary does not go from John into the car then John will shoot Mary with the gun.
Mtrans1	John told Mary that if Mary did not go from John into the car then John would shoot Mary with the gun.

Figure B.4: Kidnap Example -- Specific Explanation (Threaten)



Captive1	Mary was John's captive.
In3	In the room is in the room.
Isa48	The room is a room.
Isa49	In the room is a location.
Isa2	Mary is a person.
Isa15	John is a character.
Isa4	John is a person.
Free2	Mary was free.
Not13	Mary was not free.
Confine1	John confined Mary in the room.
At26	Mary is in the room.
Ptrans5	John moved Mary from the hotel into the room.
At23	Mary is at the hotel.
Ptrans3	John moved Mary from the car to the hotel.
Drive1	John drove Mary to the hotel in the car.
Isa41	The car is a vehicle.
Isa34	The car is a car.
At18	The car was at Mary.
In1	In the car is in the car.
At9	Mary was in the car.
At20	John was in the car.
Possess4	John has the car.
At16	John is at the hotel.
At17	The car is at the hotel.
Not7	John is not in the car.
Not8	Mary was not in the car.
Not9	The car is not at Mary.
Isa39	The hotel is a location.
Isa47	The car is a location.
Isa21	Mary is a physical object.
Isa23	Mary is an animate object.
Isa25	Mary is a character.
Isa13	John is an animate object.
At24	Mary was at the car.
Not12	Mary is not at the car.
Believe2	John believed that Mary was John's captive.

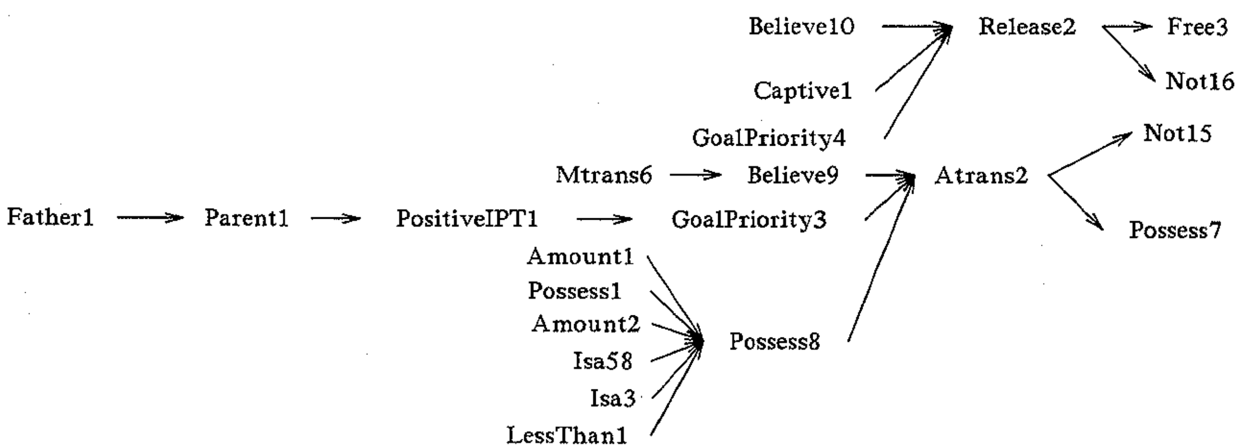
Figure B.5: Kidnap Example – Specific Explanation (Capture)



Isa54	Fred is a character.	Isa1	Fred is a person.
Isa15	John is a character.	Isa4	John is a person.
Isa2	Mary is a person.	Isa3	Millions of dollars is money.
Isa66	The $250000 is a valuable.	Isa58	The $250000 is money.
Isa48	The room is a room.	Isa49	In the room is a location.

ThemeGoalMet2	John is happy that John has the $250000.
Possess7	John has the $250000.
Bargain1	John and Fred carried out a bargain in which John released Mary and Fred gave John the $250000 at Trenos.
GoalPriority4	John wants to have the $250000 more than he wants Mary to be John's captive.
Believe5	Fred believed that Mary was John's captive.
Communicate2	John contacted Fred and told him that Mary was John's captive.
Telephone1	John called Fred and told him that Mary was John's captive.
Know2	John knows that Fred has phone number something.
Believe2	John believed that Mary was John's captive.
Capture1	John captured Mary and locked her in the room.
In3	In the room is in the room.
Free2	Mary was free.
Captive1	Mary was John's captive.
Not13	Mary was not free.
GoalPriority3	Fred wants Mary to be free more than he wants to have the $250000.
PositiveIPT1	Fred has a positive relationship with Mary.
Parent1	Fred is Mary's parent.
Father1	Fred is Mary's father.
Possess8	Fred had the $250000.
LessThan1	250000 is less than order-millions.
Amount2	The $250000 is 250000 dollars of money.
Possess1	Fred has millions of dollars.
Amount1	Millions of dollars is order-millions dollars of money.
Free3	Mary is free.
Not15	Fred does not have the $250000.
Not16	Mary is not John's captive.
Not17	Fred does not believe that Mary is John's captive.
Not18	John does not believe that Mary is John's captive.

Figure B.6: Kidnap Example – Specific Explanation (Highest-Level)



Possess7	John has the $250000.
GoalPriority4	John wants to have the $250000 more than he wants Mary to be John's captive.
GoalPriority3	Fred wants Mary to be free more than he wants to have the $250000.
PositiveIPT1	Fred has a positive relationship with Mary.
Parent1	Fred is Mary's parent.
Father1	Fred is Mary's father.
Possess8	Fred had the $250000.
LessThan1	250000 is less than order-millions.
Isa3	Millions of dollars is money.
Isa58	The $250000 is money.
Amount2	The $250000 is 250000 dollars of money.
Possess1	Fred has millions of dollars.
Amount1	Millions of dollars is order-millions dollars of money.
Captive1	Mary was John's captive.
Free3	Mary is free.
Release2	John released Mary.
Believe10	John believes that if John released Mary then Fred will give John the $250000 at Trenos.
Not16	Mary is not John's captive.
Not15	Fred does not have the $250000.
Atrans2	Fred gave John the $250000 at Trenos.
Believe9	Fred believes that if Fred gave John the $250000 at Trenos then John will release Mary.
Mtrans6	John told Fred that if Fred gave John the $250000 at Trenos then John would release Mary.

Figure B.7: Kidnap Example -- Specific Explanation (Bargain)

Residence1	Color1	At10	Name1	Name4	Gender1
Residence2	Attire1	At27	Name2	Isa59	Gender2
Isa33	Isa35	Isa38	Name3	Isa60	Gender3

Isa33	The jeans are jeans.
Isa35	In the car is a location.
Isa38	The hotel is a hotel.
Isa59	Trenos is a restaurant.
Isa60	Trenos is a location.
Gender1	Fred is male.
Gender2	Mary is female.
Gender3	John is male.
Name1	Fred's name is Fred.
Name2	Mary's name is Mary.
Name3	John's name is John.
Name4	Trenos's name is Trenos.
At10	The hotel is at the car.
At27	Trenos is at some place.
Color1	The jeans are blue.
Attire1	Mary is wearing the jeans.
Residence1	John lives in the hotel.
Residence2	John lives in the room.

Figure B.8: Unconnected Facts in the Kidnap Example

other actions. For example, if a particular instance of a Bargain involves actor X doing action A in exchange for actor Y doing action B, then the effects and preconditions of the overall Bargain include the effects and preconditions, respectively, of actions A and B. In addition, in order for the Bargain to be motivated, X should value the facts added by action B over those deleted by action A, and Y should value the facts added by A over those deleted by B. For further information, the interested reader is referred to the complete definitions of Bargain and Threaten given in section 1 of appendix C.

B.1.3. Learning CaptureBargain

After the Bargain schema is determined in the kidnapping narrative, the system detects that John has achieved a thematic goal of possessing a valuable item. Since this goal is achieved by novel combination of volitional actions by the main character, the system proceeds to learn a new schema from its explanation of how the goal was achieved. Prior to generalization, the GENESIS pruning algorithm removes a number of units from the explanation shown in Figure B.6. In addition to pruning a few Isa inferences, a number of interesting generalizations are made. The fact that Telephone was the particular specialization of Communicate used in the explanation is removed since all of its effects that support the goal are inherited from Communicate. Also, the abstraction inferences from Father to Parent and from Parent to PositiveIPT are also removed since the Father and Parent relationships only support the goal through the more abstract PositiveIPT relationship. The final generalized explanation EGGS produces is shown in Figure B.9.

The packaging of this generalized explanation into a schema does not detect any protection violations and results in the final definition shown in Figure B.10. The English summary the paraphraser produces for this schema was presented in chapter 2. English translations of the suggestions used to index this new schema were also given in the trace in chapter 2. The formal versions of these suggestions are given below:

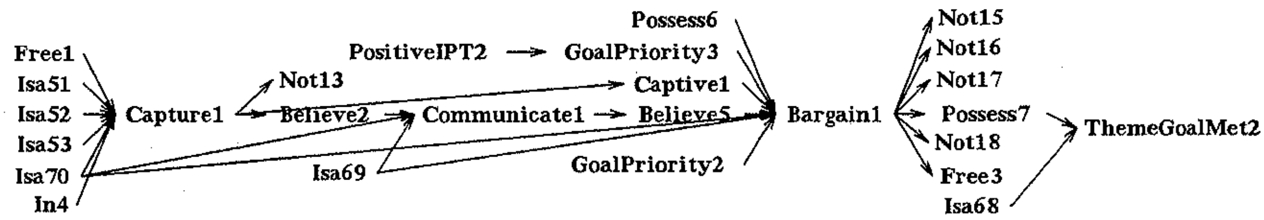
((Capture ?x55 ?b9 ?c4 ?r5) ~~> (CaptureBargain ?x55 ?a34 ?b9 ?c4 ?r5 ?y5 ?111))
((Communicate ?x55 ?a34 (Captive ?b9 ?x55 ?c4 ?r5)) ~~>
(CaptureBargain ?x55 ?a34 ?b9 ?c4 ?r5 ?y5 ?111))
((Bargain ?x55 ?a34 (Release ?x55 ?b9 ?c4 ?r5) (Atrans ?a34 ?y5 ?x55 ?111))) ~~>
(CaptureBargain ?x55 ?a34 ?b9 ?c4 ?r5 ?y5 ?111))

B.1.4. Processing the Test Narrative After Learning

When GENESIS processes the test kidnapping narrative after learning the CaptureBargain schema, the Capture schema determined from Bob's locking Alice in his basement suggests that the new schema may be useful in understanding the narrative. Bob's acquisition of $75,00 and Alice's restored freedom confirms all of the positive effects of CaptureBargain and the schema is determined. The resulting explanation for Bob's possession of the money is shown in Figure B.11. This explanation allows the system to answer the questions it could not answer previously and the CaptureBargain schema provides a reasonable paraphrase of the text.

Other previously incomprehensible narratives can also be understood using the new schema. An interesting variation of the test narrative that results in the same final explanation is the even sketchier account shown below:

Ted is Alice's husband. He won a $100000 in the lottery. Someone imprisoned Alice in a basement. Bob got $75000. Someone released Alice.

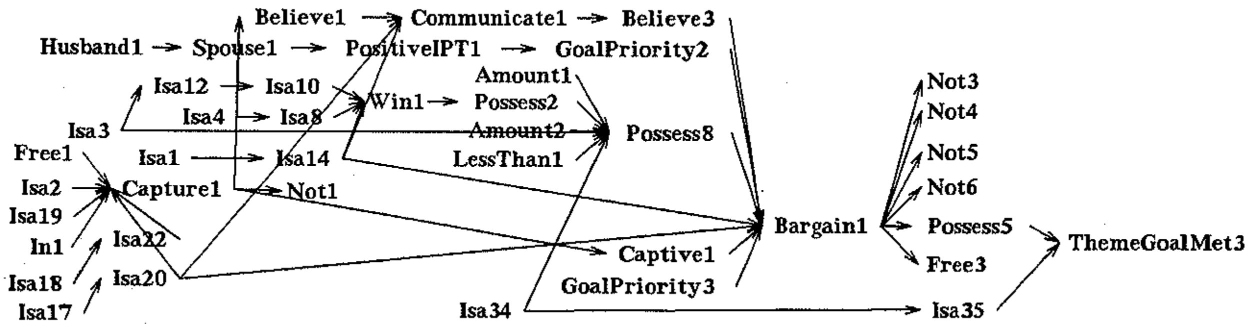


ThemeGoalMet2	?x55 is happy that ?x55 has ?y5.
Isa68	?y5 is a valuable.
Possess7	?x55 has ?y5.
Bargain1	?x55 and ?a34 carried out a bargain in which ?x55 released ?b9 and ?a34 gave ?x55 ?y5 at ?l11.
GoalPriority2	?x55 wants to have ?y5 more than he/she wants ?b9 to be ?x55's captive.
Isa69	?a34 is a character.
Isa70	?x55 is a character.
Believe5	?a34 believed that ?b9 was ?x55's captive.
Communicate1	?x55 contacted ?a34 and told him/her that ?b9 was ?x55's captive.
Believe2	?x55 believed that ?b9 was ?x55's captive.
Capture1	?x55 captured ?b9 and locked him/her in ?r5.
In4	?c4 is in ?r5.
Isa53	?r5 is a room.
Isa52	?c4 is a location.
Isa51	?b9 is a person.
Free1	?b9 was free.
Captive1	?b9 was ?x55's captive.
Not13	?b9 was not free.
GoalPriority3	?a34 wants ?b9 to be free more than he/she wants to have ?y5.
PositiveIPT2	?a34 has a positive relationship with ?b9.
Possess6	?a34 had ?y5.
Free3	?b9 is free.
Not15	?a34 does not have ?y5.
Not16	?b9 is not ?x55's captive.
Not17	?a34 does not believe that ?b9 is ?x55's captive.
Not18	?x55 does not believe that ?b9 is ?x55's captive.

Figure B.9: Kidnap Example -- Generalized Explanation

```
((CaptureBargain ?x55 ?a34 ?b9 ?c4 ?r5 ?y5 ?l11)   "?x55 kidnapped ?b9."
 (Constraint (Isa ?b9 Person) (Isa ?c4 Location) (Isa ?r5 Room) (In ?c4 ?r5)
          (Isa ?x55 Character) (Isa ?a34 Character) (Isa ?y5 Valuable))
 (Precondition (Possess ?a34 ?y5) (PositiveIPT ?a34 ?b9) (Free ?b9))
 (Motivation (GoalPriority ?x55 (Possess ?x55 ?y5) (Captive ?b9 ?x55 ?c4 ?r5)))
 (Effect (Not (Believe ?x55 (Captive ?b9 ?x55 ?c4 ?r5)))
        (Not (Believe ?a34 (Captive ?b9 ?x55 ?c4 ?r5))) (Not (Captive ?b9 ?x55 ?c4 ?r5))
        (Not (Possess ?a34 ?y5)) (Free ?b9) (Possess ?x55 ?y5))
 (Subaction (Capture ?x55 ?b9 ?c4 ?r5) (Communicate ?x55 ?a34 (Captive ?b9 ?x55 ?c4 ?r5))
          (Bargain ?x55 ?a34 (Release ?x55 ?b9 ?c4 ?r5) (Atrans ?a34 ?y5 ?x55 ?l11)))
 (Internal (Not (Free ?b9)) (Captive ?b9 ?x55 ?c4 ?r5)
          (GoalPriority ?a34 (Free ?b9) (Possess ?a34 ?y5))
          (Believe ?a34 (Captive ?b9 ?x55 ?c4 ?r5)) (Believe ?x55 (Captive ?b9 ?x55 ?c4 ?r5)))
 (Links (Antecedent (Internal 3) (Precondition 2)) (Precondition (Subaction 1) (Precondition 3))
    (Constraint (Subaction 1) (Constraint 4)) (Constraint (Subaction 1) (Constraint 3))
    (Constraint (Subaction 1) (Constraint 2)) (Constraint (Subaction 1) (Constraint 1))
    (Constraint (Subaction 1) (Constraint 5)) (Effect (Subaction 1) (Internal 5))
    (Effect (Subaction 1) (Internal 1)) (Effect (Subaction 1) (Internal 2))
    (Precondition (Subaction 2) (Internal 5)) (Constraint (Subaction 2) (Constraint 6))
    (Constraint (Subaction 2) (Constraint 5)) (Effect (Subaction 2) (Internal 4))
    (Precondition (Subaction 3) (Internal 4)) (Precondition (Subaction 3) (Internal 3))
    (Precondition (Subaction 3) (Precondition 1)) (Precondition (Subaction 3) (Internal 2))
    (Motivation (Subaction 3) (Motivation 1)) (Constraint (Subaction 3) (Constraint 6))
    (Constraint (Subaction 3) (Constraint 5)) (Effect (Subaction 3) (Effect 1))
    (Effect (Subaction 3) (Effect 2)) (Effect (Subaction 3) (Effect 3))
    (Effect (Subaction 3) (Effect 4)) (Effect (Subaction 3) (Effect 5))
    (Effect (Subaction 3) (Effect 6)))
```

Figure B.10: CaptureBargain Schema Definition



Isa35	The $75000 is a valuable.	Isa34	The $75000 is money.
Isa1	Ted is a person.	Isa14	Ted is a character.
Isa17	Bob is a person.	Isa20	Bob is a character.
Isa22	The basement is a room.	Isa18	The basement is a basement.
Isa19	In the basement is a location.	Isa2	Alice is a person.
Isa8	The lottery is a game.	Isa4	The lottery is a lottery.
Isa3	The $100000 is money.	Isa10	The $100000 is an inanimate object.
Isa12	The $100000 is a valuable.		

ThemeGoalMet3	Bob is happy that Bob has the $75000.
Possess5	Bob has the $75000.
Bargain1	Bob and Ted carried out a bargain in which Bob released Alice and Ted gave Bob the $75000.
GoalPriority3	Bob wants to have the $75000 more than he wants Alice to be Bob's captive.
Captive1	Alice was Bob's captive.
Capture1	Bob captured Alice and locked her in the basement.
In1	In the basement is in the basement.
Free1	Alice is free.
Not1	Alice was not free.
Believe1	Bob believed that Alice was Bob's captive.
Possess8	Ted had the $75000.
LessThan1	75000 is less than 100000.
Amount2	The $75000 is 75000 dollars of money.
Possess2	Ted has the $100000.
Win1	Ted won the $100000 in the lottery.
Amount1	The $100000 is 100000 dollars of money.
GoalPriority2	Ted wants Alice to be free more than he wants to have the $75000.
PositiveIPT1	Ted has a positive relationship with Alice.
Spouse1	Ted is Alice's spouse.
Husband1	Ted is Alice's husband.
Believe3	Ted believed that Alice was Bob's captive.
Communicate1	Bob contacted Ted and told him that Alice was Bob's captive.
Not3	Bob does not believe that Alice is Bob's captive.
Not4	Ted does not believe that Alice is Bob's captive.
Not5	Alice is not Bob's captive.
Not6	Ted does not have the $75000.
Free3	Alice is free.

Figure B.11: Kidnap Test Example After Learning – Specific Explanation

The processing of this narrative using CaptureBargain is basically the same as the processing of the test narrative except for the fact that Bob being the one who got the money identifies him as the kidnapper. Consequently, if after processing this narrative GENESIS is asked: "Who captured Alice?" it replies: "Bob captured Alice and locked her in the basement."

B.2. The Arson Example

In the Arson example, GENESIS learns a schema for an individual burning his own property in order to collect the insurance money. A version of this example was first used in the original GENESIS system [Mooney85b]. Before processing the narratives for the Arson example, GENESIS has schemata for burning, insuring objects, and being indemnified for the loss of property; however, it does not have a schema for arson-for-insurance.

B.2.1. Processing the Test Narrative Before Learning

In order to demonstrate the system's initial ignorance of arson-for-insurance, GENESIS is given the following test narrative:

Input: John owned a barn. He burned it. He got 40000 dollars.

Ready for questions:

>Summarize
John burned a barn.

>How did John get the money?
Cannot find sufficient cause.

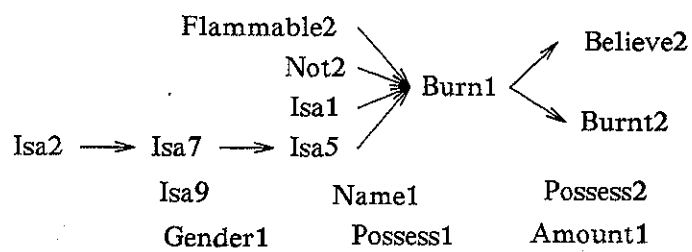
>Why did John burn the barn?
Cannot find sufficient reason.

>Who owned the barn?
John has the barn.

The parsed version of this narrative is shown in Figure B.12 and the complete causal model the system builds for this narrative is shown in Figure B.13. Without a schema for this situation or any

(Isa Person25 Person) (Gender Person25 male) (Name Person25 John)
(Isa Barn1 Barn) (Possess Person25 Barn1) (Burn Person25 Barn1)
(Isa Money5 Money) (Amount Money5 40000 dollar) (Possess Person25 Money5)

Figure B.12: Parsed Version of the Arson Test Narrative



Burnt2	The barn is burnt.
Burn1	John burned the barn.
Isa5	The barn is an inanimate object.
Isa7	The barn is a building.
Isa2	The barn is a barn.
Isa1	John is a person.
Not2	The barn was not burnt.
Flammable2	The barn is flammable.
Believe2	John believes that the barn is burnt.
Isa9	The $40000 is money.
Gender1	John is male.
Name1	John's name is John.
Possess1	John has the barn.
Possess2	John has the $40000.
Amount1	The $40000 is 40000 dollars of money.

Figure B.13: Causal Model for Arson Test Example Before Learning

mention of insurance or indemnity in the text, the system is unable to construct a satisfactory causal model; and John's possession of the money is left unconnected to his act of burning his barn. Consequently, the system is unable to answer the questions presented in the trace.

B.2.2. Processing the Learning Narrative

Next, the system is given a more detailed narrative about arson-for-insurance, which it can explain and generalize into a schema.

Input: Stan owned a warehouse. He insured it against fire for 100000 dollars. Stan burned the warehouse. He called Prudential and told them it was burnt. Prudential paid him 100000 dollars.

Thematic goal achieved: Stan is happy that Stan is insured with the insurance company for the $100000 if the warehouse is burnt.

Thematic goal achieved: Stan is happy that Stan has the $100000.
Explanation suitable for generalization. Pruning...Generalizing...Packaging...
Creating New Schema: (InsureObjectBurnIndemnify ?a3 ?o2 ?c3 ?v3 ?l4)

?a3 is a person. ?o2 is an inanimate object. ?c3 is an insurance company. ?v3 is money. ?o2 is not burnt. ?a3 has ?o2. ?a3 insures ?o2 with ?c3 for ?v3 in case it is burnt. ?o2 is flammable. ?a3 burns ?o2. ?a3 contacts ?c3 and tells it that ?o2 is burnt. ?c3 indemnifies ?a3 ?v3 for the loss of ?o2.

Having: ?a3 insured ?o2 with ?c3 for ?v3 in case it is burnt. suggest InsureObjectBurnIndemnify
Having: ?a3 burned ?o2. suggest InsureObjectBurnIndemnify
Having: ?a3 contacted ?c3 and told it that ?o2 was burnt. suggest InsureObjectBurnIndemnify

Ready for questions:

>Summarize
A warehouse was not burnt. Stan had the warehouse. Stan insured the warehouse with Prudential for $100000 in case it is burnt. The warehouse was flammable. Stan burned the warehouse. Stan contacted Prudential and told them that the warehouse was burnt. Prudential indemnified Stan the $100000 for the loss of the warehouse.

>Why did Stan insure the warehouse?
So Prudential would indemnify Stan the $100000 for the loss of the warehouse.

>Why did Stan burn the wharehouse?
=WAREHOUSE[1]

So Stan could call Prudential and could tell them that the warehouse was burnt.

>Why did Stan tell Prudential the warehouse was burnt?
So Prudential would indemnify Stan the $100000 for the loss of the warehouse.

>Why did Prudential pay Stan the money?
Because Prudential believed that the warehouse was burnt and because Stan was insured with Prudential for the $100000 if the warehouse was burnt.

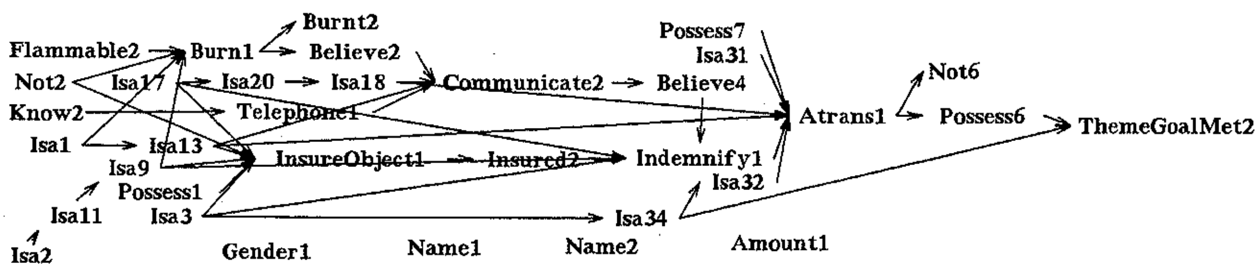
>Why did Prudential believe the warehouse was burnt?
Because Stan called Prudential and told them that the warehouse was burnt.

The parsed version of this narrative is shown in Figure B.14 and the explanation the system constructs for why Stan has the money is shown in Figure B.15. This explanation can be summarized as follows. Stan insuring the warehouse has the effect of making it insured and Stan burning the barn has the effect of making it burnt. When Stan calls Prudential it suggests a Telephone schema, which is determined when Stan tells them that the barn is burnt. Prudential's action of giving Stan $100,000 is recognized as an instance of the Indemnify schema. Since Indemnify is a specialization of Atrans (see the GENESIS Action hierarchy in chapter 9) and since the actor of an Indemnify is constrained to be an insurance company (which Prudential is known to be), the Atrans is specialized to an Indemnify. This transformation makes use of one of Lytinen's [Lytinen84] parsing rules,

```
(Isa Person26 Person) (Gender Person26 male) (Name Person26 Stan) (Isa Warehouse1 Warehouse)
(Possess Person26 Warehouse1) (Isa Money6 Money) (Amount Money6 100000 dollar)
(InsureObject Person26 (Burnt Warehouse1) Money6 ?COMPANY2) (Burn Person26 Warehouse1)
(Isa InsuranceCo1 InsuranceCo) (Name InsuranceCo1 Prudential)
(DialTelephone Person26 InsuranceCo1 ?NUMBER135)
(Mtrans Person26 (Burnt Warehouse1) InsuranceCo1)
(Atrans InsuranceCo1 Money6 Person26 ?AT85)
```

Figure B.14: Parsed Version of the Arson Learning Narrative

[1]The INTERLISP spelling corrector [Teitelman83] is used to correct such typos.



ThemeGoalMet2	Stan is happy that Stan has the $100000.
Isa34	The $100000 is a valuable.
Isa3	The $100000 is money.
Possess6	Stan has the $100000.
Atrans1	Prudential gave Stan the $100000.
Indemnify1	Prudential indemnified Stan the $100000 for the loss of the warehouse.
Insured2	Stan is insured with Prudential for the $100000 if the warehouse is burnt.
InsureObject1	Stan insured the warehouse with Prudential for the $100000 in case it is burnt.
Isa17	Prudential is an insurance company.
Isa9	The warehouse is an inanimate object.
Isa11	The warehouse is a building.
Isa2	The warehouse is a warehouse.
Isa13	Stan is a character.
Isa1	Stan is a person.
Not2	The warehouse was not burnt.
Possess1	Stan has the warehouse.
Believe4	Prudential believes that the warehouse is burnt.
Communicate2	Stan contacted Prudential and told them that the warehouse was burnt.
Telephone1	Stan called Prudential and told them that the warehouse was burnt.
Know2	Stan knows that Prudential has phone number something.
Isa18	Prudential is a character.
Isa20	Prudential is a company.
Believe2	Stan believes that the warehouse is burnt.
Burn1	Stan burned the warehouse.
Flammable2	The warehouse is flammable.
Burnt2	The warehouse is burnt.
Isa31	Some place is a location.
Isa32	The $100000 is an inanimate object.
Possess7	Prudential had the $100000.
Not6	Prudential does not have the $100000.
Gender1	Stan is male.
Name1	Stan's name is Stan.
Name2	Prudential's name is Prudential.
Amount1	The $100000 is 100000 dollars of money.

Figure B.15: Arson Example -- Specific Explanation

the original definition of which is given below:

> SLOT-FILLER-SPECIALIZATION RULE: If a slot of concept A is filled by concept B, and B is the prototypical filler for that slot of concept C, and concept C ISA-A concept A, then change the representation of concept A to concept C (p.224).

In the example, A is the Atrans, B is InsuranceCo, and C is Indemnify. The preconditions and motivations of the Indemnify include that Prudential believe that the warehouse is burnt (which matches an effect of Stan's Telephone action) and that the warehouse is insured against fire for $100,000 (which matches an effect of Stan's InsureObject action). The resulting explanation as shown in Figure B.15 allows GENESIS to answer the questions about this narrative presented in the trace.

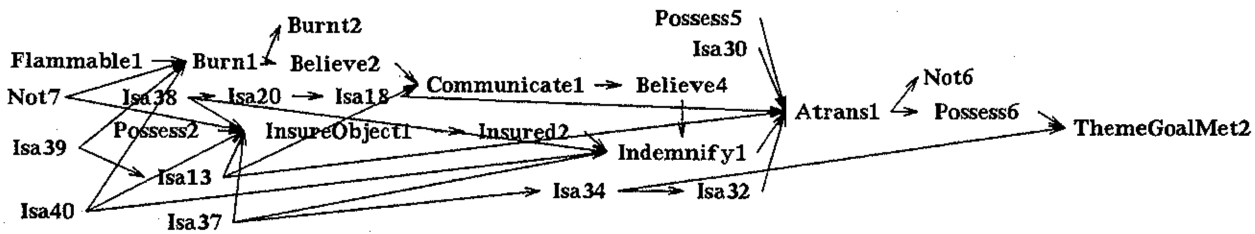
B.2.3. Learning InsureObjectBurnIndemnify

After the effects of the Indemnify action are added, GENESIS detects that Stan has achieved the thematic goal of possessing a valuable item in a novel way. The only action not volitionally performed by Stan (the main character) is Prudential's Indemnify action and this action is motivated by the fact that they believe the warehouse is burnt, which is in turn supported by Stan's Burn and Communicate actions. Consequently the explanation satisfies all of the learning criteria and the system proceeds to learn a new schema from its explanation for how Stan's goal was achieved. Prior to generalization, several units are pruned from the explanation shown in Figure B.15. In addition to pruning the fact that the burned item is a warehouse, the specialization of Communicate to Telephone is also pruned as it was in the kidnapping example. The final generalized explanation EGGS produces for this example is shown in Figure B.16.

The packaging of this generalized explanation into a schema detects an interesting protection violation and imposes an additional ordering constraint to prevent it. The system detects that the Burn action can potentially clobber a precondition of the InsureObject action, namely that the item being insured cannot already be in the state that it is being insured against (being burnt in this case). As is clear from the graph in Figure B.16, the structure of the explanation itself does not impose a temporal ordering on these two actions. To avoid the violation, the system adds the following constraint to the new schema:

> InsureObject1 should proceed Burn1 when T or else Burnt2 will clobber Not7.

The final definition for the schema learned from this example is given in Figure B.17.



ThemeGoalMet2	?a3 is happy that ?a3 has ?v3.
Isa34	?v3 is a valuable.
Isa37	?v3 is money.
Possess6	?a3 has ?v3.
Atrans1	?c3 gave ?a3 ?v3 at ?l4.
Indemnify1	?c3 indemnified ?a3 ?v3 for the loss of ?o2.
Insured2	?a3 is insured with ?c3 for ?v3 if ?o2 is burnt.
InsureObject1	?a3 insured ?o2 with ?c3 for ?v3 in case it is burnt.
Isa38	?c3 is an insurance company.
Isa40	?o2 is an inanimate object.
Isa13	?a3 is a character.
Isa39	?a3 is a person.
Not7	?o2 is not burnt.
Possess2	?a3 has ?o2.
Believe4	?c3 believes that ?o2 is burnt.
Communicate1	?a3 contacted ?c3 and told them that ?o2 was burnt.
Isa18	?c3 is a character.
Isa20	?c3 is a company.
Believe2	?a3 believes that ?o2 is burnt.
Burn1	?a3 burned ?o2.
Flammable1	?o2 is flammable.
Burnt2	?o2 is burnt.
Isa30	?l4 is a location.
Isa32	?v3 is an inanimate object.
Possess5	?c3 had ?v3.
Not6	?c3 does not have ?v3.

Figure B.16: Arson Example – Generalized Explanation

```
((InsureObjectBurnIndemnify ?a3 ?o2 ?c3 ?v3 ?l4)   "?a3 did something to ?o2."
 (Constraint (Isa ?l4 Location) (Isa ?a3 Person) (Isa ?o2 Inanimate)
          (Isa ?c3 InsuranceCo) (Isa ?v3 Money))
 (Precondition (Possess ?c3 ?v3) (Flammable ?o2) (Possess ?a3 ?o2) (Not (Burnt ?o2)))
 (Effect (Insured (Burnt ?o2) ?a3 ?v3 ?c3) (Believe ?a3 (Burnt ?o2)) (Burnt ?o2)
        (Believe ?c3 (Burnt ?o2)) (Not (Possess ?c3 ?v3)) (Possess ?a3 ?v3))
 (Subaction (InsureObject ?a3 (Burnt ?o2) ?v3 ?c3) (Burn ?a3 ?o2)
          (Communicate ?a3 ?c3 (Burnt ?o2)) (Indemnify ?c3 ?v3 ?a3 (Burnt ?o2))
          (Atrans ?c3 ?v3 ?a3 ?l4))
 (Internal (Isa ?v3 Inanimate) (Isa ?c3 Character) (Isa ?c3 Company) (Isa ?a3 Character)
          (Isa ?v3 Valuable))
 (Links (Antecedent (Internal 1) (Internal 5)) (Antecedent (Internal 2) (Internal 3))
     (Antecedent (Internal 3) (Constraint 4)) (Antecedent (Internal 4) (Constraint 2))
     (Antecedent (Internal 5) (Constraint 5)) (Precondition (Subaction 1) (Precondition 4))
     (Precondition (Subaction 1) (Precondition 3)) (Constraint (Subaction 1) (Constraint 4))
     (Constraint (Subaction 1) (Constraint 5)) (Constraint (Subaction 1) (Constraint 3))
     (Constraint (Subaction 1) (Internal 4)) (Effect (Subaction 1) (Effect 1))
     (Precondition (Subaction 2) (Precondition 4)) (Precondition (Subaction 2) (Precondition 2))
     (Constraint (Subaction 2) (Constraint 3)) (Constraint (Subaction 2) (Constraint 2))
     (Effect (Subaction 2) (Effect 2)) (Effect (Subaction 2) (Effect 3))
     (Precondition (Subaction 3) (Effect 2)) (Constraint (Subaction 3) (Internal 2))
     (Constraint (Subaction 3) (Internal 4)) (Effect (Subaction 3) (Effect 4))
     (Motivation (Subaction 4) (Effect 1)) (Motivation (Subaction 4) (Effect 4))
     (Constraint (Subaction 4) (Constraint 3)) (Constraint (Subaction 4) (Constraint 5))
     (Constraint (Subaction 4) (Constraint 4)) (Antecedent (Subaction 5) (Subaction 4))
     (Precondition (Subaction 5) (Precondition 1)) (Constraint (Subaction 5) (Constraint 1))
     (Constraint (Subaction 5) (Internal 4)) (Constraint (Subaction 5) (Internal 1))
     (Constraint (Subaction 5) (Internal 2)) (Effect (Subaction 5) (Effect 5))
     (Effect (Subaction 5) (Effect 6)))
 (Ordering ((Before (Subaction 1) (Subaction 2)) T))
```

Figure B.17: InsureObjectBurnIndemnify Schema Definition

B.2.4. Processing the Test Narrative After Learning

Below is the trace produced by GENESIS when processing the Arson test narrative for the second time:

Input: John owned a barn. He burned it. He got 40000 dollars.

Thematic goal achieved: John is happy that John is insured with the insurance company for the $40000 if the barn is burnt.

Thematic goal achieved: John is happy that John has the $40000.

Ready for questions:

>Summarize

A barn was not burnt. John had the barn. John insured the barn with an insurance company for $40000 in case it is burnt. The barn was flammable. John burned the barn. John contacted the insurance company and told them that the barn was burnt. The insurance company indemnified John the $40000 for the loss of the barn.

>How did John get the money?

The insurance company indemnified John the $40000 for the loss of the barn.

>Why did the company pay John the money?

Because John was insured with the insurance company for the $40000 if the barn was burnt and because the insurance company believed that the barn was burnt.

>Why did John burn the barn?

So John could contact the insurance company and could tell them that the barn was burnt.

>Why did John insure the barn

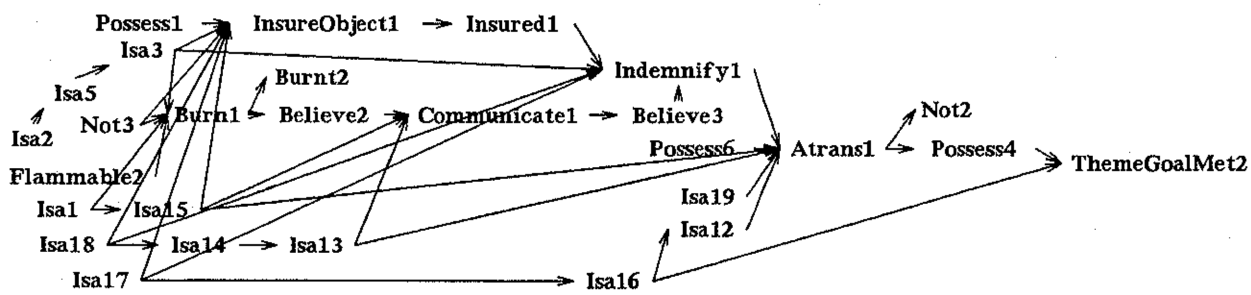
So the insurance company would indemnify John the $40000 for the loss of the barn.

>Why did the company believe the barn was burnt?

Because John contacted the insurance company and told them that the barn was burnt.

This time, John's action of burning his barn suggests the InsureObjectBurnIndemnify schema, which is determined when its effect that John acquires money is confirmed. The explanation constructed for why John has the money is shown in Figure B.18. This explanation allows it to answer questions it was previously unable to answer and the new schema provides a reasonable paraphrase of the narrative.

It is also interesting to look at some variations of the test narrative that either do or do not invoke the InsureObjectBurnIndemnify schema. For example, the following narrative does *not* invoke the new schema:



ThemeGoalMet2	John is happy that John has the $40000.
Isa16	The $40000 is a valuable.
Isa17	The $40000 is money.
Possess4	John has the $40000.
Atrans1	The insurance company gave John the $40000.
Isa13	The insurance company is a character.
Isa14	The insurance company is a company.
Isa18	The insurance company is an insurance company.
Isa12	The $40000 is an inanimate object.
Isa15	John is a character.
Isa1	John is a person.
Isa19	Some place is a location.
Possess6	The insurance company had the $40000.
Indemnify1	The insurance company indemnified John the $40000 for the loss of the barn.
Believe3	The insurance company believes that the barn is burnt.
Communicate1	John contacted the insurance company and told them that the barn was burnt.
Believe2	John believes that the barn is burnt.
Burn1	John burned the barn.
Isa3	The barn is an inanimate object.
Isa5	The barn is a building.
Isa2	The barn is a barn.
Flammable2	The barn is flammable.
Not3	The barn was not burnt.
Burnt2	The barn is burnt.
Insured1	John is insured with the insurance company for the $40000 if the barn is burnt.
InsureObject1	John insured the barn with the insurance company for the $40000 in case it is burnt.
Possess1	John has the barn.
Not2	The insurance company does not have the $40000.

Figure B.18: Arson Test Example After Learning -- Specific Explanation

Input: John owned a barn. He insured it against fire for 100000 dollars. The barn is burnt. Allstate gave John 100000 dollars.

Thematic goal achieved: John is happy that John is insured with the insurance company for the $100000 if the barn is burnt.
Thematic goal achieved: John is happy that John has the $100000.

Ready for questions:

>Why is the barn burnt?
Cannot find sufficient reason.

>Who burned the barn?
Question refers to unknown event, person, place, or thing.

>Summarize
John insured a barn with Allstate for $100000 in case it is burnt.
Allstate indemnified John the $100000 for the loss of the barn.

Although the action of insuring an item against fire was made to suggest the new schema, the act of insuring an item already achieves a known thematic goal of wanting security against the loss of one's valuable items. Consequently, since actions that are already explained by a thematic goal are not used to suggest schemata (see section 10.2), InsureObjectBurnIndemnify is not suggested in this situation.

The following variation of the test narrative; however, *does* invoke the new schema:

Gene owned a restaurant. Someone burned the restaurant. Allstate gave Gene 100000 dollars.

If after processing this narrative GENESIS is asked: "Who burned the restaurant?" it replies: "Gene burned the restaurant." This time, the fact that someone intentionally burned the restaurant does suggest the new arson-for-insurance schema since the system has no other explanation for this action. The fact that Gene got the money identifies him as the agent and determines the schema. Of course, since understanding is a subjective process, this conclusion could be wrong.

B.3. The Solicit Example

Since all of the other schemata GENESIS has learned involve illegal means of acquiring wealth, it only seemed fair to also give the system the opportunity to learn a schema for upholding the law. The system was therefore given an example from which it could learn a schema for a police-officer impersonating a prostitute in order to arrest potential customers. Before processing the narratives in this example, the system has schemata for prostitution customers soliciting sexual favors and for police-officers arresting law-breakers; however, it does not have a schema for "solicitation entrapment."

B.3.1. Processing the Test Narrative Before Learning

First the system is presented with the following test narrative:

Input: Alice was a policewoman. Stan told Alice if she had sex with him then he would give her $75. Stan went to jail.

Ready for questions:

>Summarize
Stan solicited Alice's sexual favors for $75. Stan was put in a jail.

>Why did Stan tell Alice if she had sex with him then he would give her money?
Because Stan believed that Alice was a prostitute and because Stan was horny.

>Why did Stan go to jail?
Because Stan was under arrest for something.

>Why was Stan under arrest?
Cannot find sufficient reason.

The parsed version of this narrative is presented in Figure B.19 and the causal model constructed for the narrative is shown in Figure B.20. The fact that Stan asked Alice to have sex with him for money suggests and determines a Solicit schema[1] and the fact that Stan went to jail suggests and determines an Incarcerate schema. However, without a schema for the overall situation or any

```
(Isa Person34 Person) (Name Person34 Alice) (Gender Person34 female)
(Occupation Person34 police-officer) (Isa Person35 Person) (Gender Person35 male)
(Name Person35 Stan) (Isa Money10 Money) (Amount Money10 75 dollar)
(Mtrans Person35 (Implies (Coitus Person34 Person35)
                          (Atrans Person35 Money10 Person34 ?AT110)) Person34)
(Isa Jail1 Jail) (Isa Location7 Location) (At Jail1 Location7)
(Ptrans Person35 Person35 ?FROM1179 Location7)
```

Figure B.19: Parsed Version of the Solicit Test Example

explicit mention of the arrest, the system is unable to connect Stan's incarceration with his solicitation. The system is therefore unable to appropriately answer some of the questions shown in the trace.

B.3.2. Processing the Learning Narrative

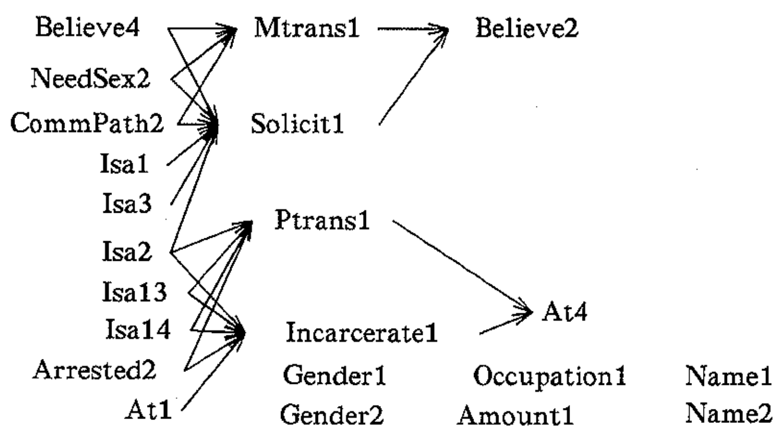
Next the system is given the following learning narrative:

Input: Jane is a policewoman. She dressed in a short red skirt and went to a corner. Bob approached the corner and told her if she had sex with him then he would give her $50. Jane arrested Bob for soliciting. Bob is Mary's husband and he told her that Jane entrapped him.

Thematic goal achieved: Jane is happy that Bob is under arrest for soliciting Jane's sexual favors for the $50.
Explanation suitable for generalization. Pruning...Generalizing...Packaging...
Creating New Schema: (DressSolicitArrest ?b9 ?c2 ?s2 ?s4 ?14 ?a9 ?c3)

?b9 is not equal to ?a9. ?b9 is a person. ?c2 is an apparel. ?b9 has ?c2. ?b9 puts on ?c2. ?s2 is a location. ?14 is a corner. ?b9 is at ?s2. ?b9 goes from ?s2 to ?14. ?a9 is a person. ?s4 is a location. ?a9 is at ?s4. ?a9 goes from ?s4 to ?14. ?c3 is money. ?a9 is horny. ?c2 is sexy. ?a9 solicits ?b9's sexual favors for ?c3. ?b9 is a police-officer. ?b9 arrests ?a9 for soliciting ?b9's sexual favors for ?c3.

[1]The Solcit schema represents a customer offering money for sexual favors as opposed to the more common English interpretation of the word "solicit" referring to a prostitute offering her services for money.



At4	Stan is at the jail.
Incarcerate1	Stan was put in the jail.
Arrested2	Stan is under arrest for something.
At1	The jail is at Stan.
Isa14	The jail is a location.
Isa13	The jail is a jail.
Isa2	Stan is a person.
Ptrans1	Stan went to the jail.
Believe2	Alice believes that Stan solicited Alice's sexual favors for the $75.
Solicit1	Stan solicited Alice's sexual favors for the $75.
NeedSex2	Stan is horny.
Believe4	Stan believes that Alice is a prostitute.
Isa3	The $75 is money.
Isa1	Alice is a person.
CommPath2	Stan has a communication path to Alice.
Mtrans1	Stan told Alice that if Alice had sex with Stan then Stan would give Alice the $75.
Name1	Alice's name is Alice.
Name2	Stan's name is Stan.
Gender1	Alice is female.
Gender2	Stan is male.
Occupation1	Alice is a police-officer.
Amount1	The $75 is 75 dollars of money.

Figure B.20: Causal Model for Solcit Test Example Before Learning

Having: ?b9 put on ?c2. suggest DressSolicitArrest

Having: ?a9 solicited ?b9's sexual favors for ?c3. suggest DressSolicitArrest

Having: ?b9 arrested ?a9 for soliciting ?b9's sexual favors for ?c3. suggest DressSolicitArrest

Unknown word '?b9 entrap ?a9' refers to DressSolicitArrest

Ready for questions:

>Summarize

Jane had a skirt. Jane put on the skirt. Jane was at some place. Jane went to a corner. Bob was at some place. Bob went to the corner. Bob was horny. The skirt was sexy. Bob solicited Jane's sexual favors for $50. Jane was a police-officer. Jane arrested Bob for soliciting Jane's sexual favors for the $50.

>Why did Jane put on the short skirt?

So Bob would solicit Jane's sexual favors for the $50.

>Why did Jane go to the corner?

So Bob would solicit Jane's sexual favors for the $50.

>Why did Bob go to the corner?

So Bob could solicit Jane's sexual favors for the $50.

>Why did Bob solicit Jane's sexual favors?

Because Bob believed that Jane was a prostitute and because Bob was horny.

>Why did Bob believe Jane was a prostitute?

Because Bob went to the corner, because Jane went to the corner, because the skirt was short and because Jane put on the skirt.

>Why did Jane arrest Bob?

Because Jane believed that Bob solicited Jane's sexual favors for the $50 and because Jane was a police-officer.

>Why did Jane believe that Bob solicited her sexual favors?

Because Bob solicited Jane's sexual favors for the $50.

The parsed version of this narrative is shown in Figure B.21 and the explanation constructed for why Bob is under arrest is shown in Figure B.22. This explanation can be summarized as follows. Jane's actions of putting on a short skirt and going to the corner has the effect of her being on the corner dressed in a short skirt. Bob's action of going to the corner results in a communication path between Bob and Jane and enables him to tell her that he would give her money if she had sex with him. The latter action is recognized as a Solicit schema, which in order to be motivated requires that Bob believe that Jane is a prostitute. This belief is inferred from the fact that she is wearing a short skirt and standing on a corner. Jane's act of arresting Bob requires several motivations. She must believe that Bob committed an illegal act (which matches an effect of the Solicit since soliciting is known to be illegal) and she must be a police-officer (which matches an input fact). The resulting explanation as shown in Figure B.22 allows GENESIS to answer the questions about this narrative presented in the trace.

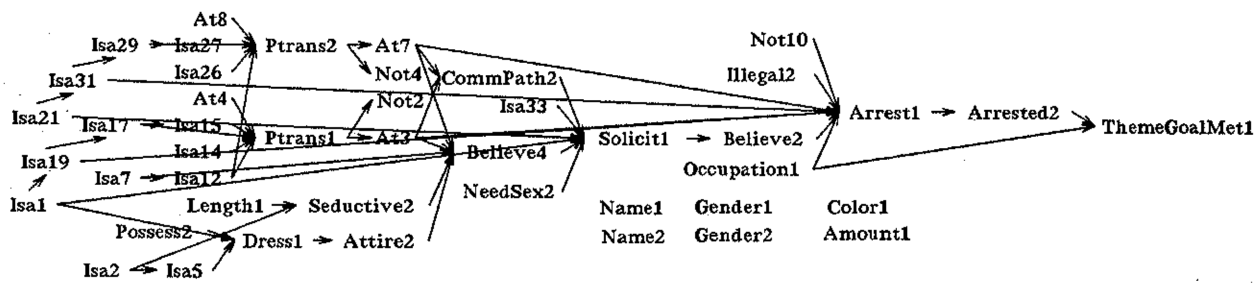
B.3.3. Learning DressSolicitArrest

After the effects of the Arrest action are added, GENESIS detects that a police-officer has achieved her thematic goal of having someone under arrest. The only action not volitionally executed by Alice (the main character) is Bob's Solicit action and this action is motivated by the fact that he believes Alice is a prostitute, which in turn is supported by Alice's actions of putting on a

---

```
(Isa Person38 Person) (Name Person38 Jane) (Gender Person38 female)
(Occupation Person38 police-officer) (Length Skirt1 short) (Color Skirt1 red)
(Isa Skirt1 Skirt) (Dress Person38 Skirt1) (Isa Corner1 Corner)
(Ptrans Person38 Person38 ?FROM1182 Corner1) (Isa Person39 Person)
(Gender Person39 male) (Name Person39 Bob) (Ptrans Person39 Person39 ?FROM1208 Corner1)
(Isa Money11 Money) (Amount Money11 50 dollar)
(Mtrans Person39 (Implies (Coitus Person38 Person39)
                 (Atrans Person39 Money11 Person38 ?AT134)) Person38)
(Arrest Person38 Person39 (Solicit Person39 ?SUBJECT97 ?FOR3)) (Isa Person43 Person)
(Name Person43 Mary) (Gender Person43 female) (Husband Person39 Person43)
(Mtrans Person39 (?Action2 Person38 Person39) Person43)
```

Figure B.21: Parsed Version of the Solicit Narrative



Name1	Jane's name is Jane.	Name2	Bob's name is Bob.
Gender1	Jane is female.	Gender2	Bob is male.
Color1	The skirt is red.	Amount1	The $50 is 50 dollars of money.
Isa5	The skirt is an apparel.	Isa2	The skirt is a skirt.
Isa15	Jane is a physical object.	Isa17	Jane is an animate object.
Isa19	Jane is a character.	Isa1	Jane is a person.
Isa27	Bob is a physical object.	Isa29	Bob is an animate object.
Isa31	Bob is a character.	Isa21	Bob is a person.

ThemeGoalMet1	Jane is happy that Bob is under arrest for soliciting Jane's sexual favors for the $50.
Occupation1	Jane is a police-officer.
Arrested2	Bob is under arrest for soliciting Jane's sexual favors for the $50.
Arrest1	Jane arrested Bob for soliciting Jane's sexual favors for the $50.
Believe2	Jane believes that Bob solicited Jane's sexual favors for the $50.
Solicit1	Bob solicited Jane's sexual favors for the $50.
NeedSex2	Bob is horny.
Believe4	Bob believes that Jane is a prostitute.
Attire2	Jane is wearing the skirt.
Dress1	Jane put on the skirt.
Possess2	Jane has the skirt.
Seductive2	The skirt is sexy.
Length1	The skirt is short.
At3	Jane is at the corner.
Ptrans1	Jane went to the corner.
Isa12	The corner is a location.
Isa7	The corner is a corner.
Isa14	Jane is a location.
At4	Jane was at some place.
Not2	Jane is not at some place.
At7	Bob is at the corner.
Ptrans2	Bob went to the corner.
Isa26	Bob is a location.
At8	Bob was at some place.
Not4	Bob is not at some place.
Isa33	The $50 is money.
CommPath2	Bob has a communication path to Jane.
Illegal2	It is illegal for Bob to solicit Jane's sexual favors for the $50.
Not10	Jane is not equal to Bob.

Figure B.22: Solicit Example – Specific Explanation

short skirt and going to a corner. Since all of the learning criteria are met, the system proceeds to learn a new schema from its explanation of how Alice's goal was achieved. Prior to generalization, the only pruning that is performed is the removal of the facts requiring that the police-officer's seductive clothing be a short skirt. These facts are removed since the system knows other ways of inferring that an apparel is seductive (e.g tight jeans). The generalized explanation for this example is given in Figure B.23 and the final definition for the learned schema is given in Figure B.24. The observant reader will notice that the police-officer is not constrained to be female. This is because the rule for inferring that someone believes that a person is a prostitute does not require that person to be female. If this additional antecedent were added to the rule, the resulting schema would be sexist and require a female police-officer.

In addition to learning a new schema from this narrative, GENESIS also learns the meaning of a new word. The last sentence of the narrative causes it to associate the unknown word "entrap" with the newly learned schema. This word is learned in a manner directly analogous to the learning of the word "kidnap" as discussed in chapter .

B.3.4. Processing the Test Narrative After Learning

Below is the trace produced by GENESIS when processing the test narrative after having learned the new schema:

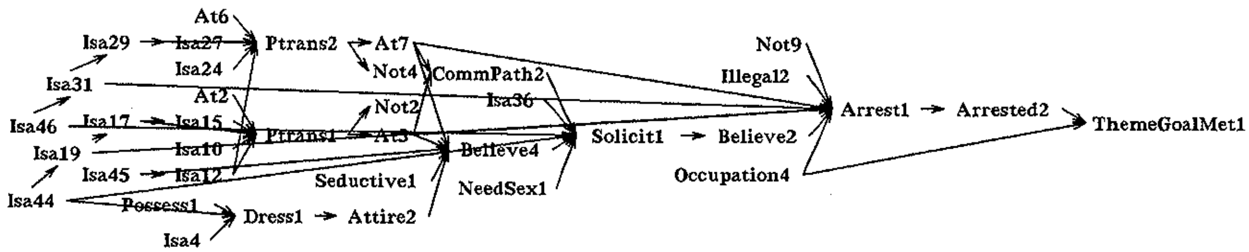
Input: Alice was a policewoman. Stan told Alice if she had sex with him then he would give her $75. Stan went to jail.

Thematic goal achieved: Alice is happy that Stan is under arrest for soliciting Alice's sexual favors for the $75.

Ready for questions:

>Summarize

Alice had an apparel. Alice put on the apparel. Alice was at some place. Alice went to a corner. Stan was at some place. Stan went to the corner. Stan was horny. The apparel was sexy. Stan solicited Alice's sexual favors for $75. Alice was a police-officer. Alice arrested Stan for soliciting Alice's sexual favors for the $75.



ThemeGoalMet1	?b9 is happy that ?a9 is under arrest for soliciting ?b9's sexual favors for ?c3.
Occupation4	?b9 is a police-officer.
Arrested2	?a9 is under arrest for soliciting ?b9's sexual favors for ?c3.
Arrest1	?b9 arrested ?a9 for soliciting ?b9's sexual favors for ?c3.
Believe2	?b9 believes that ?a9 solicited ?b9's sexual favors for ?c3.
Solicit1	?a9 solicited ?b9's sexual favors for ?c3.
NeedSex1	?a9 is horny.
Believe4	?a9 believes that ?b9 is a prostitute.
Attire2	?b9 is wearing ?c2.
Dress1	?b9 put on ?c2.
Isa4	?c2 is an apparel.
Isa44	?b9 is a person.
Possess1	?b9 has ?c2.
Seductive1	?c2 is sexy.
At3	?b9 is at ?l4.
Ptrans1	?b9 went from ?s2 to ?l4.
Isa12	?l4 is a location.
Isa45	?l4 is a corner.
Isa10	?s2 is a location.
Isa15	?b9 is a physical object.
Isa17	?b9 is an animate object.
Isa19	?b9 is a character.
At2	?b9 was at ?s2.
Not2	?b9 is not at ?s2.
At7	?a9 is at ?l4.
Ptrans2	?a9 went from ?s4 to ?l4.
Isa24	?s4 is a location.
Isa27	?a9 is a physical object.
Isa29	?a9 is an animate object.
Isa31	?a9 is a character.
Isa46	?a9 is a person.
At6	?a9 was at ?s4.
Not4	?a9 is not at ?s4.
Isa36	?c3 is money.
CommPath2	?a9 has a communication path to ?b9.
Illegal2	It is illegal for ?a9 to solicit ?b9's sexual favors for ?c3.
Not9	?b9 is not equal to ?a9.

Figure B.23: Solicit Example -- Generalized Explanation

```
(DressSolicitArrest ?b9 ?c2 ?s2 ?s4 ?14 ?a9 ?c3)  "?b9 entrapped ?a9."
 (Constraint (Not (EQUAL ?b9 ?a9)) (Isa ?c3 Money) (Isa ?a9 Person) (Isa ?s4 Location)
          (Isa ?s2 Location) (Isa ?14 Corner) (Isa ?b9 Person) (Isa ?c2 Apparel))
 (Precondition (At ?a9 ?s4) (At ?b9 ?s2) (Seductive ?c2) (Possess ?b9 ?c2) (NeedSex ?a9))
 (Motivation (Occupation ?b9 police-officer))
 (Effect (Attire ?b9 ?c2) (Not (At ?b9 ?s2)) (At ?b9 ?14) (Not (At ?a9 ?s4)) (At ?a9 ?14)
        (Believe ?b9 (Solicit ?a9 ?b9 ?c3)) (Arrested ?a9 (Solicit ?a9 ?b9 ?c3) ?b9))
 (Subaction (Dress ?b9 ?c2) (Ptrans ?b9 ?b9 ?s2 ?14) (Ptrans ?a9 ?a9 ?s4 ?14)
         (Solicit ?a9 ?b9 ?c3) (Arrest ?b9 ?a9 (Solicit ?a9 ?b9 ?c3)))
 (Internal (Illegal (Solicit ?a9 ?b9 ?c3)) (CommPath ?a9 ?b9)
          (Believe ?a9 (Occupation ?b9 prostitute)) (Isa ?a9 PhysicalObject)
          (Isa ?a9 Animate) (Isa ?a9 Character) (Isa ?b9 PhysicalObject)
          (Isa ?b9 Animate) (Isa ?b9 Character) (Isa ?14 Location))
 (Links (Antecedent (Internal 2) (Effect 5)) (Antecedent (Internal 2) (Effect 3))
     (Antecedent (Internal 3) (Effect 1)) (Antecedent (Internal 3) (Precondition 3))
     (Antecedent (Internal 3) (Effect 3)) (Antecedent (Internal 3) (Effect 5))
     (Antecedent (Internal 3) (Constraint 6)) (Antecedent (Internal 4) (Internal 5))
     (Antecedent (Internal 5) (Internal 6)) (Antecedent (Internal 6) (Constraint 3))
     (Antecedent (Internal 7) (Internal 8)) (Antecedent (Internal 8) (Internal 9))
     (Antecedent (Internal 9) (Constraint 7)) (Antecedent (Internal 10) (Constraint 6))
     (Precondition (Subaction 1) (Precondition 4)) (Constraint (Subaction 1) (Constraint 8))
     (Constraint (Subaction 1) (Constraint 7)) (Effect (Subaction 1) (Effect 1))
     (Precondition (Subaction 2) (Precondition 2)) (Constraint (Subaction 2) (Internal 10))
     (Constraint (Subaction 2) (Constraint 5)) (Constraint (Subaction 2) (Internal 7))
     (Constraint (Subaction 2) (Internal 8)) (Effect (Subaction 2) (Effect 2))
     (Effect (Subaction 2) (Effect 3)) (Precondition (Subaction 3) (Precondition 1))
     (Constraint (Subaction 3) (Internal 10)) (Constraint (Subaction 3) (Constraint 4))
     (Constraint (Subaction 3) (Internal 4)) (Constraint (Subaction 3) (Internal 5))
     (Effect (Subaction 3) (Effect 4)) (Effect (Subaction 3) (Effect 5))
     (Precondition (Subaction 4) (Internal 2)) (Motivation (Subaction 4) (Precondition 5))
     (Motivation (Subaction 4) (Internal 3)) (Constraint (Subaction 4) (Constraint 2))
     (Constraint (Subaction 4) (Constraint 7)) (Constraint (Subaction 4) (Constraint 3))
     (Effect (Subaction 4) (Effect 6)) (Precondition (Subaction 5) (Effect 3))
     (Precondition (Subaction 5) (Effect 5)) (Motivation (Subaction 5) (Motivation 1))
     (Motivation (Subaction 5) (Effect 6)) (Constraint (Subaction 5) (Internal 1))
     (Constraint (Subaction 5) (Constraint 1)) (Constraint (Subaction 5) (Internal 6))
       (Constraint (Subaction 5) (Internal 9)) (Effect (Subaction 5) (Effect 7)))
```

Figure B.24: DressSolicitArrest Schema Definition

>Why did Stan go to jail?

Because Stan was under arrest for soliciting Alice's sexual favors for the $75.

>Why was Stan under arrest?

Because Alice entrapped Stan.

>Who arrested Stan?

Alice arrested Stan for soliciting Alice's sexual favors for the $75.

>Why did Alice arrest Stan?

Because Alice was a police-officer and because Alice believed that Stan solicited Alice's sexual favors for the $75.

>Why did Stan tell Alice if she had sex with him then he would give her money?

Because Stan believed that Alice was a prostitute and because Stan was horny.

>Why did Stan believe that Alice was a prostitute?

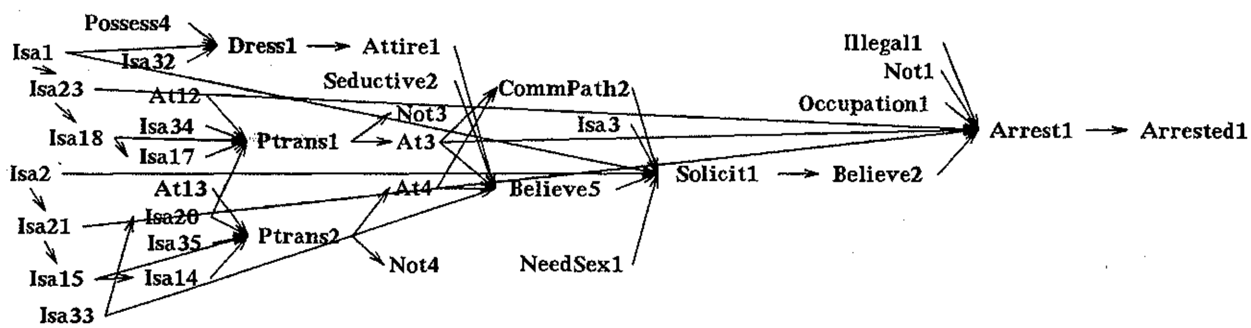
Because Alice entrapped Stan and because the apparel was sexy.

This time, Stan's action of soliciting Alice suggests the new DressSolicitArrest schema. This schema is determined when its effect of Stan being under arrest for solicitation matches a fact added as a precondition of his incarceration. The system's explanation for why Stan is under arrest is shown in Figure B.25.

The fact that the test narrative mentions that Alice is a police-officer is not crucial to the understanding of this narrative. For example, the following narrative is also understood using the learned schema:

> Alice was at a corner wearing tight blue jeans. Stan told Alice if she had sex with him then he would give her $75. Stan went to jail.

If after processing this narrative GENESIS is asked: "Who arrested Stan?," it replies: "Alice arrested Stan for soliciting Alice's sexual favors for the $75." Because the schema suggestion and determination procedures are based primarily on confirming that certain actions have occurred, the presence of precondition facts in the narrative, such as the fact that Alice is a police-officer, does



Arrested1	Stan is under arrest for soliciting Alice's sexual favors for the $75.
Arrest1	Alice arrested Stan for soliciting Alice's sexual favors for the $75.
Believe2	Alice believes that Stan solicited Alice's sexual favors for the $75.
Solicit1	Stan solicited Alice's sexual favors for the $75.
NeedSex1	Stan is horny.
Believe5	Stan believes that Alice is a prostitute.
Isa33	The corner is a corner.
At4	Stan is at the corner.
Ptrans2	Stan went to the corner.
Isa15	Stan is an animate object.
Isa21	Stan is a character.
Isa2	Stan is a person.
Isa14	Stan is a physical object.
Isa35	Stan is a location.
Isa20	The corner is a location.
At13	Stan was at some place.
Not4	Stan is not at some place.
At3	Alice is at the corner.
Ptrans1	Alice went to the corner.
Isa18	Alice is an animate object.
Isa23	Alice is a character.
Isa1	Alice is a person.
Isa17	Alice is a physical object.
Isa34	Alice is a location.
At12	Alice was at some place.
Not3	Alice is not at some place.
Seductive2	The apparel is sexy.
Attire1	Alice is wearing the apparel.
Dress1	Alice put on the apparel.
Isa32	The apparel is an apparel.
Possess4	Alice has the apparel.
Isa3	The $75 is money.
CommPath2	Stan has a communication path to Alice.
Occupation1	Alice is a police-officer.
Not1	Alice is not equal to Stan.
Illegal1	It is illegal for Stan to solicit Alice's sexual favors for the $75.

Figure B.25: Solicit Test Example After Learning -- Specific Explanation

not effect these processes. Of course, a better understanding system would probably consider the number of a schema's preconditions that are known to be satisfied and use this information to help choose between competing interpretations of the text.

GENESIS can also use its conjectured meaning for the word "entrap" to help understand narratives that it would otherwise find incomprehensible. For example, in the following narrative, the presence of the word "entrap" is the only thing that suggests the DressSolicitArrest schema.

Input: Jan entrapped Mike. Mike went to jail.

Thematic goal achieved: Jan is happy that Mike is under arrest for soliciting Jan's sexual favors for the money.

Ready for questions:

>Summarize
Jan had an apparel. Jan put on the apparel. Jan was at some place. Jan went to a corner. Mike was at some place. Mike went to the corner. Mike was horny. The apparel was sexy. Mike solicited Jan's sexual favors for money. Jan was a police-officer. Jan arrested Mike for soliciting Jan's sexual favors for the money.

Of course, the English word "entrap" does not actually refer specifically to "solicitation-entrapment;" however, given the system's current experience with this word, its interpretation of this narrative is reasonable.

APPENDIX C

INITIAL GENESIS KNOWLEDGE

C.1. Action Schema Definitions

```
((Die ?a)  "?a died."
 (Constraint (Isa ?a Person))
 (Effect (Dead ?a)))

((Inherit ?a ?o ?d)  "?a inherited ?d's ?o."
 (Constraint (Isa ?a Character) (Isa ?d Person) (Isa ?o Inanimate))
 (Precondition (Heir ?a ?d) (Possess ?d ?o) (Dead ?d))
 (Effect (Possess ?a ?o) (Not (Possess ?d ?o))))

((Win ?a ?o ?c)  "?a won ?o in ?c."
 (Constraint (Isa ?a Character) (Isa ?o Inanimate) (Isa ?c Game))
 (Effect (Possess ?a ?o)))

((Aim ?a ?o ?s)  "?a aimed ?o at ?s."
 (Constraint (Isa ?a Person) (Isa ?o PhysicalObject) (Isa ?s PhysicalObject))
 (Precondition (LosPath ?a ?s) (Possess ?a ?o))
 (Effect (PointingAt ?a ?o ?s)))

((Arrest ?a ?b (?d ?b . ?r))  "?a arrested ?b for (?d ?b . ?r)."
 (Constraint (Isa ?a Character) (Isa ?b Character) (Not (EQUAL ?a ?b)) (Illegal (?d ?b . ?r)))
 (Precondition (At ?b ?l) (At ?a ?l))
 (Motivation (Believe ?a (?d ?b . ?r)) (Occupation ?a police-officer))
 (Effect (Arrested ?b (?d ?b . ?r) ?a)))

((Atrans ?a ?o ?t ?l)  "?a gave ?t ?o at ?l."
 (Constraint (Isa ?a Character) (Isa ?o Inanimate) (Isa ?t Character) (Isa ?l Location))
 (Precondition (Possess ?a ?o))
 (Effect (Possess ?t ?o) (Not (Possess ?a ?o)))
 (Suggest ((Atrans ?a ?o ?t ?l) ~~> (Poison ?a ?t ?o))))

((Indemnify ?a ?o ?t (?s ?i))  "?a indemnified ?t ?o for the loss of ?i."
 (Constraint (Isa ?a InsuranceCo) (Isa ?o Money) (Isa ?i Inanimate))
 (Motivation (Believe ?a (?s ?i)) (Insured (?s ?i) ?t ?o ?a)))

((Bargain ?a ?b (?p ?a . ?r) (?d ?b . ?l))
 "?a and ?b carried out a bargain in which (?p ?a . ?r) and (?d ?b . ?l)."
 (Constraint (Isa ?a Character) (Isa ?b Character))
 (Precondition (*Pointer* (?p ?a . ?r) Precondition) (*Pointer* (?d ?b . ?l) Precondition)
               (GoalPriority ?b (*Pointer* (?p ?a . ?r) PosEffect) (*Pointer* (?d ?b . ?l) NegEffect))
               (Believe ?b (*Pointer* (?p ?a . ?r) Precondition)))
 (Motivation (GoalPriority ?a (*Pointer* (?d ?b . ?l) PosEffect) (*Pointer* (?p ?a . ?r) NegEffect)))
```

```
(Effect (*Pointer* (?d ?b . ?l) PosEffect) (*Pointer* (?p ?a . ?r) PosEffect)
          (Not (*Pointer* (?d ?b . ?l) NegEffect)) (Not (*Pointer* (?p ?a . ?r) NegEffect))
          (Not (Believe ?b (*Pointer* (?p ?a . ?r) NegEffect)))
          (Not (Believe ?a (*Pointer* (?p ?a . ?r) NegEffect))))
(Subaction (Mtrans ?a (Implies (?d ?b . ?l) (?p ?a . ?r)) ?b) (?d ?b . ?l) (?p ?a . ?r))
(Internal (Believe ?b (Implies (?d ?b . ?l) (?p ?a . ?r))) (Believe ?a (Implies (?p ?a . ?r) (?d ?b . ?l))))
(Links (Effect (Subaction 2) (Effect 1)) (Effect (Subaction 3) (Effect 2))
        (Effect (Subaction 2) (Effect 3)) (Effect (Subaction 3) (Effect 4))
        (Precondition (Subaction 2) (Precondition 2)) (Effect (Subaction 1) (Internal 1))
        (Precondition (Subaction 3) (Precondition 1)) (Motivation (Subaction 2) (Internal 1))
        (Motivation (Subaction 2) (Precondition 3)) (Motivation (Subaction 3) (Motivation 1))
        (Motivation (Subaction 3) (Internal 2))))

((Burn ?a ?o)  "?a burned ?o."
 (Constraint (Isa ?a Person) (Isa ?o Inanimate))
 (Precondition (Flammable ?o) (Not (Burnt ?o)))
 (Effect (Burnt ?o) (Believe ?a (Burnt ?o))))

((Capture ?a ?b ?c ?r)  "?a captured ?b and locked it in ?r."
 (Constraint (Isa ?a Character) (Isa ?b Person) (Isa ?c Location) (Isa ?r Room) (In ?c ?r))
 (Precondition (Free ?b))
 (Effect (Captive ?b ?a ?c ?r) (Not (Free ?b)) (Believe ?a (Captive ?b ?a ?c ?r)))
 (Subaction (Ptrans ?a ?b ?d ?c) (Ptrans ?a ?b ?s ?d) (Confine ?a ?b ?c ?r))
 (Internal (At ?b ?d) (At ?b ?c))
 (Links (Constraint (Subaction 3) (Constraint 4)) (Constraint (Subaction 3) (Constraint 5))
        (Constraint (Subaction 3) (Constraint 1)) (Constraint (Subaction 1) (Constraint 1))
        (Constraint (Subaction 1) (Constraint 3)) (Constraint (Subaction 2) (Constraint 1))
        (Effect (Subaction 2) (Internal 1)) (Precondition (Subaction 1) (Internal 1))
        (Effect (Subaction 1) (Internal 2)) (Precondition (Subaction 3) (Internal 2))
        (Precondition (Subaction 3) (Precondition 1)) (Effect (Subaction 3) (Effect 1))
        (Effect (Subaction 3) (Effect 2)) (Effect (Subaction 3) (Effect 3))))

((Communicate ?a ?o ?i)  "?a contacted ?o and told it that ?i."
 (Constraint (Isa ?a Character) (Isa ?o Character))
 (Precondition (Believe ?a ?i))
 (Effect (Believe ?o ?i))
 (Subaction (Mtrans ?a ?i ?o))
 (Links (Precondition (Subaction 1) (Precondition 1)) (Effect (Subaction 1) (Effect 1))
        (Constraint (Subaction 1) (Constraint 1)) (Constraint (Subaction 1) (Constraint 2))))

((Telephone ?a ?o ?i)  "?a called ?o and told it that ?i."
 (Precondition (Know ?a (HasPhoneNumber ?o ?n)))
 (Subaction (DialTelephone ?a ?o ?n) (AnswerTelephone ?o ?a) (Talk ?a ?i ?o))
 (Internal (PhoneRinging ?o ?a) (CommPath ?a ?o))
 (Links (Precondition (Subaction 1) (Precondition 1)) (Effect (Subaction 1) (Internal 1))
        (Precondition (Subaction 2) (Internal 1)) (Effect (Subaction 2) (Internal 2))
        (Precondition (Subaction 3) (Internal 2)))
 (Abstractions ((Subaction 3) Communicate (Subaction 1))))

((Confine ?a ?b ?c ?r)  "?a confined ?b ?c."
 (Suggest ((Confine ?a ?b ?c ?r) ~~> (Capture ?a ?b ?c ?r))))
```

```
((DialTelephone ?a ?o ?n)  "?a called ?o."
 (Suggest ((DialTelephone ?a ?o ?n) ~~> (Telephone ?a ?o ?i)))))

((Dress ?a ?c)  "?a put on ?c."
 (Constraint (Isa ?a Person) (Isa ?c Apparel))
 (Precondition (Possess ?a ?c))
 (Effect (Attire ?a ?c)))

((Incarcerate ?a ?o ?j ?d)  "?o was put in ?j."
 (Constraint (Isa ?o Person) (Isa ?j Jail) (Isa ?d Location) (At ?j ?d))
 (Motivation (Arrested ?o ?v ?p))
 (Effect (At ?o ?d))
 (Subaction (Ptrans ?o ?o ?s ?d))
 (Links (Constraint (Subaction 1) (Constraint 1)) (Constraint (Subaction 1) (Constraint 2))
        (Constraint (Subaction 1) (Constraint 3)) (Motivation (Subaction 1) (Motivation 1))
        (Effect (Subaction 1) (Effect 1))))

((InsureObject ?a (?s ?o) ?v ?c)  "?a insured ?o with ?c for ?v in case (?s ?o)."
 (Constraint (Isa ?a Character) (Isa ?o Inanimate) (Isa ?v Money) (Isa ?c InsuranceCo))
 (Precondition (Possess ?a ?o) (Not (?s ?o)))
 (Effect (Insured (?s ?o) ?a ?v ?c)))

((Mtrans ?a ?i ?o)  "?a told ?o that ?i."
 (Suggest ((Mtrans ?a (Implies (Coitus ?b ?a) (Atrans ?a ?o ?b ?1)) ?b) ~~> (Solicit ?a ?b ?o))
          ((Mtrans ?a (Implies (?p ?b . ?r) (?d ?a . ?1)) ?b) ~~> (Bargain ?a ?b (?d ?a . ?1) (?p ?b . ?r)))
          ((Mtrans ?a (Implies (Not (?p ?b . ?r)) (?d ?a . ?1)) ?b) ~~>
               (Threaten ?a ?b (?d ?a . ?1) (?p ?b . ?r)))))

((Murder ?a ?v)  "?a murdered ?v."
 (Constraint (Isa ?a Character) (Isa ?v Person))
 (Effect (Dead ?v))
 (Subaction (GenesisAction ?a) (Die ?v))
 (Internal (State ?v) (State ?o) (BadHealth ?v))
 (Links (Effect (Subaction 1) (Internal 1)) (Antecedent (Internal 3) (Internal 1))
        (Antecedent (Internal 3) (Internal 2)) (Precondition (Subaction 2) (Internal 3))
        (Effect (Subaction 2) (Effect 1))))

((Poison ?a ?v ?p)  "?a poisoned ?v with ?p."
 (Constraint (Poisonous ?p) (Isa ?p Food))
 (Precondition (Possess ?a ?p) (NeedSustenance ?v))
 (Effect (Inside ?p ?v) (Not (Possess ?a ?p)) (Not (Possess ?v ?p)))
 (Subaction (Atrans ?a ?p ?v ?1) (Ingest ?v ?p) (Die ?v))
 (Internal (Possess ?v ?p) (BadHealth ?v))
 (Links (Precondition (Subaction 1) (Precondition 1)) (Effect (Subaction 1) (Internal 1))
        (Effect (Subaction 1) (Effect 2)) (Constraint (Subaction 2) (Constraint 2))
        (Precondition (Subaction 2) (Internal 1)) (Motivation (Subaction 2) (Precondition 2))
        (Effect (Subaction 2) (Effect 1)) (Effect (Subaction 2) (Effect 3))
        (Antecedent (Internal 2) (Constraint 1)) (Antecedent (Internal 2) (Effect 1))
        (Precondition (Subaction 3) (Internal 2)))
 (Abstractions ((Internal 2) Murder (Internal 3)) ((Internal 1) Murder (Internal 1))
              ((Effect 1) Murder (Internal 2)) ((Subaction 3) Murder (Subaction 2))
              ((Subaction 1) Murder (Subaction 1))))
```

```
((ShootToKill ?a ?v ?g)  "?a shot ?v and killed it."
 (Constraint (Isa ?g Gun))
 (Precondition (Possess ?a ?g))
 (Effect (Shot ?v))
 (Subaction (Shoot ?a ?v ?g) (Die ?v))
 (Internal (BadHealth ?v))
 (Links (Precondition (Subaction 1) (Precondition 1)) (Effect (Subaction 1) (Effect 1))
        (Antecedent (Internal 1) (Effect 1)) (Precondition (Subaction 2) (Internal 1)))
 (Abstractions ((Effect 1) Murder (Internal 1)) ((Effect 1) Murder (Internal 2))
               ((Internal 1) Murder (Internal 3)) ((Subaction 2) Murder (Subaction 2))
               ((Subaction 1) Murder (Subaction 1))))

((Ptrans ?a ?o ?s ?d)  "?a moved ?o from ?s to ?d."
 (Constraint (Isa ?a Animate) (Isa ?o PhysicalObject) (Isa ?s Location) (Isa ?d Location))
 (Precondition (At ?o ?s))
 (Effect (At ?o ?d) (Not (At ?o ?s)))
 (Suggest (((Ptrans ?a ?o ?s ?d) (Isa ?j Jail) (At ?j ?d)) ~~> (Incarcerate ?p ?o ?j ?d))))

((Drive ?a ?o ?s ?d ?c)  "?a drove ?o from ?s to ?d in ?c."
 (Constraint (Isa ?c Vehicle))
 (Precondition (Possess ?a ?c) (At ?a ?i) (At ?o ?i) (In ?i ?c) (At ?c ?s))
 (Effect (At ?a ?d) (At ?c ?d) (Not (At ?a ?i)) (Not (At ?o ?i)) (Not (At ?c ?s))))

((Release ?a ?b ?c ?r)  "?a released ?b."
 (Precondition (Captive ?b ?a ?c ?r))
 (Effect (Free ?b) (Not (Captive ?b ?a ?c ?r))))

((Shoot ?a ?v ?g)  "?a shot ?v with ?g."
 (Constraint (Isa ?g Gun) (Isa ?a Person))
 (Precondition (Possess ?a ?g)(PointingAt ?a ?g ?v))
 (Effect (Shot ?v))
 (Suggest ((Shoot ?a ?v ?g) ~~> (ShootToKill ?a ?v ?g))))

((Solicit ?a ?b ?o)  "?a solicited ?b's sexual favors for ?o."
 (Constraint (Isa ?a Person) (Isa ?b Person) (Isa ?o Money))
 (Precondition (CommPath ?a ?b))
 (Motivation (Believe ?a (Occupation ?b prostitute)) (NeedSex ?a))
 (Effect (Believe ?b (Solicit ?a ?b ?o)))
 (Subaction (Mtrans ?a (Implies (Coitus ?b ?a) (Atrans ?a ?o ?b ?1)) ?b))
 (Links (Motivation (Subaction 1) (Motivation 1)) (Motivation (Subaction 1) (Motivation 2))
        (Precondition (Subaction 1) (Precondition 1)) (Effect (Subaction 1) (Effect 1))))

((Threaten ?a ?b (?p ?a . ?r) (?d ?b . ?1))  "?a threatened (?p ?a . ?r) unless (?d ?b . ?1)."
 (Constraint (Isa ?a Character) (Isa ?b Character))
 (Precondition (*Pointer* (?p ?a . ?r) Precondition) (*Pointer* (?d ?b . ?1) Precondition)
               (Goal ?b (Not (*Pointer* (?p ?a . ?r) Effect))))
 (Effect (*Pointer* (?d ?b . ?1) Effect) (Not (*Pointer* (?d ?b . ?1) NegEffect)))
 (Subaction (Mtrans ?a (Implies (Not (?d ?b . ?1)) (?p ?a . ?r)) ?b) (?d ?b . ?1))
 (Internal (Believe ?b (Implies (Not (?d ?b . ?1)) (?p ?a . ?r))))
 (Links (Precondition (Subaction 2) (Precondition 2)) (Motivation (Subaction2) (Precondition 3))
        (Motivation (Subaction 2) (Internal 1)) (Effect (Subaction 1) (Internal 1))
        (Effect (Subaction 2) (Effect 1)) (Effect (Subaction 2) (Effect 2))))
```

C.2. Inference Rules

```
((Seductive ?x)  "?x was sexy."
 (Brules ((Seductive ?x) <== (Isa ?x Skirt) (Length ?x short))
         ((Seductive ?x) <== (Isa ?x Jeans) (Fit ?x tight))))

((CommPath ?a ?b)  "?a had a communication path to ?b."
 (Brules ((CommPath ?a ?b) <== (At ?b ?l) (At ?a ?l))))

((Heir ?x ?y)  "?x was ?y's heir."
 (Brules ((Heir ?x ?y) <== (Spouse ?x ?y))
         ((Heir ?x ?y) <== (Parent ?y ?x))))

((Illegal ?a)  "It was illegal for ?a."
 (Brules ((Illegal (Solicit ?a ?b ?c)) <==)))

((LosPath ?a ?b)  "?a had a line of sight path to ?b."
 (Brules ((LosPath ?a ?b) <== (At ?a ?l) (At ?b ?l))))

((Believe ?x ?p)  "?x believed that ?p."
 (Brules ((Believe ?x (Occupation ?y prostitute)) <== (Isa ?l Corner)(At ?x ?l) (At ?y ?l)
                 (Seductive ?c) (Attire ?y ?c))))

((GoalPriority ?a ?d ?f)  "?a wanted ?d more than it wanted ?f."
 (Brules ((GoalPriority ?a (Free ?b) (Possess ?a ?o)) <== (PositiveIPT ?a ?b))))

((Spouse ?a ?b)  "?a was ?b's spouse."
 (Brules ((Spouse ?a ?b) <== (Spouse ?b ?a))
         ((Spouse ?a ?b) <== (Married ?a ?b))))

((Possess ?a ?x)  "?a had ?x."
 (Brules ((Possess ?a ?x) <== (Amount ?y ?g ?u) (Possess ?a ?y) (Amount ?x ?l ?u)
              (Isa ?x ?t) (Isa ?y ?t) (LessThan ?l ?g))
         ((Possess ?a ?x) <== (Holding ?a ?x))))

((ThemeGoalMet ?x ?g)  "?x was happy that ?g."
 (Brules ((ThemeGoalMet ?x (Possess ?x ?y)) <== (Possess ?x ?y) (Isa ?y Valuable))
         ((ThemeGoalMet ?a (Insured ?s ?a ?v ?c)) <== (Insured ?s ?a ?v ?c))
         ((ThemeGoalMet ?a (Arrested ?b ?d ?a)) <== (Arrested ?b ?d ?a)
                 (Occupation ?a police-officer))))
```

APPENDIX D

DETAILED GENESIS TRACE FOR THE MURDER EXAMPLE

```
10_(ProcessStory MURDER-T)

Input: (Martha is Gene's mother and is a millionaire)

Input: (Isa Person1 Person)
    Add state: Isa1

Input: (Gender Person1 female)
    Add state: Gender1

Input: (Name Person1 Martha)
    Add state: Name1

Input: (Isa Person2 Person)
    Add state: Isa2

Input: (Gender Person2 male)
    Add state: Gender2

Input: (Name Person2 Gene)
    Add state: Name2

Input: (Isa Money1 Money)
    Add state: Isa3

Input: (Amount Money1 order-millions dollar)
    Add state: Amount1

Input: (Mother Person1 Person2)
    Add state: Mother1
    Inferring Parent1: (Parent Person1 Person2)
    Inferring PositiveIPT1: (PositiveIPT Person1 Person2)

Input: (Possess Person1 Money1)
    Add state: Possess1

Input: (He shot her and she died)

Input: (Shoot Person2 Person1 ?INSTR6)

Suggesting: ShootToKill1:(ShootToKill Person2 Person1 ?INSTR6)

Suggesting: Murder1:(Murder Person2 Person1)

Found suggested Shoot2:(Shoot Person2 Person1 ?INSTR6)
   matches Shoot1:(Shoot Person2 Person1 ?INSTR6)
  Assuming suggested: Possess2:(Possess Person2 ?INSTR6)
  Assuming suggested: Shot2:(Shot Person1)
  Assuming suggested: BadHealth1:(BadHealth Person1)
  Assuming suggested: BadHealth2:(BadHealth Person1)
  Assuming suggested: State2:(State ?o1)
  Assuming suggested: State1:(State Person1)
```

```
  Assuming suggested: GenesisAction1:(GenesisAction Person2)

Input: (Die Person1)

Found suggested Die2:(Die Person1)
   matches Die3:(Die Person1)
  Assuming suggested: Dead1:(Dead Person1)
  Assuming suggested: Die1:(Die Person1)
Determining: ShootToKill1: (ShootToKill Person2 Person1 ?INSTR6)

    Looking for Precondition:(Possess Person2 ?INSTR6)
     Unable to infer (Possess Person2 ?INSTR6)
     Precondition not found,
       Assuming Possess3: (Possess Person2 ?INSTR6)

    Looking for Constraint:(Isa ?INSTR6 Gun)
     Constraint not found,
       Assuming Isa7: (Isa ?INSTR6 Gun)

    Effects:
      Shot2: (Shot Person1)
         Murder2: (Murder Person2 Person1)inferrred from:
            ShootToKill1: (ShootToKill Person2 Person1 ?INSTR6)

Determining: Murder1: (Murder Person2 Person1)

    Looking for Constraint:(Isa Person1 Person)
      Found Isa1:(Isa Person1 Person)

    Looking for Constraint:(Isa Person2 Character)
     Isa8: (Isa Person2 Character)inferrred from:
        Isa2: (Isa Person2 Person)

    Effects:
      Dead1: (Dead Person1)
         GenesisAction2: (GenesisAction Person2 . ?x9)inferrred from:
            Shoot2: (Shoot Person2 Person1 ?INSTR6)
         State3: (State Person1)inferrred from:
            Shot2: (Shot Person1)
         State4: (State Person1)inferrred from:
            Shot2: (Shot Person1)

Input: (Gene got $1000000)

Input: (Isa Money2 Money)
    Add state: Isa10

Input: (Amount Money2 1000000 dollar)
    Add state: Amount2

Input: (Possess Person2 Money2)
    Add state: Possess4
     Unable to infer (Isa Money2 Gun)

Ready for questions:

>
11_(ProcessStory MURDER)
```

Input: (Claudius was Agrippina's husband and owned an estate)

Input: (Isa Person1 Person)
Add state: Isa1

Input: (Gender Person1 male)
Add state: Gender1

Input: (Name Person1 Claudius)
Add state: Name1

Input: (Isa Person2 Person)
Add state: Isa2

Input: (Gender Person2 female)
Add state: Gender2

Input: (Name Person2 Agrippina)
Add state: Name2

Input: (Isa Estate1 Estate)
Add state: Isa3

Input: (Husband Person1 Person2)
Add state: Husband1
Inferring Spouse1: (Spouse Person1 Person2)
Inferring PositiveIPT1: (PositiveIPT Person1 Person2)

Input: (Possess Person1 Estate1)
Add state: Possess1

Input: (Agrippina gave him a poisonous mushroom and he died)

Input: (Poisonous Mushroom1)
Add state: Poisonous1

Input: (Isa Mushroom1 Mushroom)
Add state: Isa4

Input: (Atrans Person2 Mushroom1 Person1 ?AT6)

Found Poisonous1: (Poisonous Mushroom1)
Isa5: (Isa Mushroom1 Food)inferrred from:
Isa4: (Isa Mushroom1 Mushroom)

Suggesting: Poison1:(Poison Person2 Person1 Mushroom1)

Suggesting: Murder1:(Murder Person2 Person1)

Found suggested Atrans2:(Atrans Person2 Mushroom1 Person1 ?12)
matches Atrans1:(Atrans Person2 Mushroom1 Person1 ?AT6)
Assuming suggested: Possess3:(Possess Person2 Mushroom1)
Assuming suggested: Not2:(Not (Possess Person2 Mushroom1))
Assuming suggested: Possess4:(Possess Person1 Mushroom1)
Assuming suggested: State1:(State Person1)
Assuming suggested: GenesisAction1:(GenesisAction Person2)

Input: (Die Person1)

```
Found suggested Die2:(Die Person1)
   matches Die3:(Die Person1)
  Assuming suggested: BadHealth2:(BadHealth Person1)
  Assuming suggested: State2:(State ?o2)
  Assuming suggested: Inside1:(Inside Mushroom1 Person1)
  Assuming suggested: Ingest1:(Ingest Person1 Mushroom1)
  Assuming suggested: NeedSustenance1:(NeedSustenance Person1)
  Assuming suggested: Isa7:(Isa Mushroom1 Food)
  Assuming suggested: Not3:(Not (Possess Person1 Mushroom1))
  Assuming suggested: BadHealth1:(BadHealth Person1)
  Assuming suggested: Poisonous2:(Poisonous Mushroom1)
  Assuming suggested: Dead1:(Dead Person1)
  Assuming suggested: Die1:(Die Person1)
Determining: Poison1: (Poison Person2 Person1 Mushroom1)

   Looking for Constraint:(Isa Mushroom1 Food)
     Found Isa5:(Isa Mushroom1 Food)

   Looking for Constraint:(Poisonous Mushroom1)
     Found Poisonous1:(Poisonous Mushroom1)

   Looking for Precondition:(NeedSustenance Person1)
    Unable to infer (NeedSustenance Person1)
    Precondition not found,
      Assuming NeedSustenance2: (NeedSustenance Person1)

   Looking for Precondition:(Possess Person2 Mushroom1)
    Unable to infer (Possess Person2 Mushroom1)
    Precondition not found,
      Assuming Possess5: (Possess Person2 Mushroom1)

   Effects:
     Inside1: (Inside Mushroom1 Person1)
     Not2: (Not (Possess Person2 Mushroom1))
       Not2 deletes Possess5: (Possess Person2 Mushroom1)
     Not3: (Not (Possess Person1 Mushroom1))
       Not3 deletes Possess4: (Possess Person1 Mushroom1)
         Murder2: (Murder Person2 Person1)inferrred from:
           Poison1: (Poison Person2 Person1 Mushroom1)

Determining: Murder1: (Murder Person2 Person1)

   Looking for Constraint:(Isa Person1 Person)
     Found Isa1:(Isa Person1 Person)

   Looking for Constraint:(Isa Person2 Character)
    Isa10: (Isa Person2 Character)inferrred from:
      Isa2: (Isa Person2 Person)

   Effects:
     Dead1: (Dead Person1)
       GenesisAction2: (GenesisAction Person2 . ?x13)inferrred from:
         Atrans2: (Atrans Person2 Mushroom1 Person1 ?12)
       State3: (State Person1)inferrred from:
         Possess4: (Possess Person1 Mushroom1)
       State4: (State Mushroom1)inferrred from:
         Inside1: (Inside Mushroom1 Person1)

Input: (She inherited the estate)
```

```
Input: (Inherit Person2 Estate1 ?FROM3)
    Create action: Inherit1
      Found  Isa1: (Isa Person1 Person)

    Looking for Precondition:(Dead ?FROM3)
      Found Dead1:(Dead Person1)

    Looking for Precondition:(Possess Person1 Estate1)
      Found Possess1:(Possess Person1 Estate1)

    Looking for Constraint:(Isa Person1 Person)
      Found Isa1:(Isa Person1 Person)

    Looking for Constraint:(Isa Person2 Character)
      Found Isa10:(Isa Person2 Character)

    Looking for Precondition:(Heir Person2 Person1)
     Heir2: (Heir Person2 Person1)inferrred from:
        Spouse4: (Spouse Person2 Person1)inferrred from:
           Spouse1: (Spouse Person1 Person2)
    Inferring PositiveIPT2: (PositiveIPT Person2 Person1)

    Looking for Constraint:(Isa Estate1 Inanimate)
     Isa15: (Isa Estate1 Inanimate)inferrred from:
        Isa17: (Isa Estate1 Valuable)inferrred from:
           Isa3: (Isa Estate1 Estate)

    Effects:
      Possess9: (Possess Person2 Estate1)
      Not6: (Not (Possess Person1 Estate1))
       Not6 deletes Possess1: (Possess Person1 Estate1)
     ThemeGoalMet1: (ThemeGoalMet Person2 (Possess Person2 Estate1)) inferrred from:
        Isa17: (Isa Estate1 Valuable)
        Possess9: (Possess Person2 Estate1)

Thematic goal achieved: ThemeGoalMet1:(ThemeGoalMet Person2 (Possess Person2 Estate1))

Explanation suitable for generalization.

Pruning..
Pruning unit:Isa17 ((Isa Estate1 Valuable)
               <==
               (Isa Estate1 Estate))
Pruning unit:Isa10 ((Isa Person2 Character)
               <==
               (Isa Person2 Person))
Pruning unit:Isa5 ((Isa Mushroom1 Food)
              <==
              (Isa Mushroom1 Mushroom))
Pruning unit:Murder2 ((Murder Person2 Person1)
                <==
                (Poison Person2 Person1 Mushroom1))
Pruning unit:Spouse1 ((Spouse Person1 Person2)
                <==
                (Husband Person1 Person2))
Pruning unit:Spouse4 ((Spouse Person2 Person1)
                <==
                (Spouse Person1 Person2))
Pruning unit:Heir2 ((Heir Person2 Person1)
```

```
                <==
                (Spouse Person2 Person1))
Generalizing...
Packaging...

In general Not6 deletes Possess8 when T
Creating New Schema: (MurderInherit ?a5 ?o4 ?d1)

Schema summary:(MurderInherit ?a5 ?o4 ?d1)
?a5 is a character.
?d1 is a person.
?a5 murders ?d1.
?o4 is a valuable.
?d1 has ?o4.
?a5 is ?d1's heir.
?a5 inherits ?d1's ?o4.

Having: (Murder ?a5 ?d1) suggest MurderInherit

Ready for questions:

>
12_(ProcessStory MURDER-T)

Input: (Martha is Gene's mother and is a millionaire)

Input: (Isa Person1 Person)
    Add state: Isa1

Input: (Gender Person1 female)
    Add state: Gender1

Input: (Name Person1 Martha)
    Add state: Name1

Input: (Isa Person2 Person)
    Add state: Isa2

Input: (Gender Person2 male)
    Add state: Gender2

Input: (Name Person2 Gene)
    Add state: Name2

Input: (Isa Money1 Money)
    Add state: Isa3

Input: (Amount Money1 order-millions dollar)
    Add state: Amount1

Input: (Mother Person1 Person2)
    Add state: Mother1
    Inferring Parent1: (Parent Person1 Person2)
    Inferring PositiveIPT1: (PositiveIPT Person1 Person2)

Input: (Possess Person1 Money1)
    Add state: Possess1

Input: (He shot her and she died)
```

```
Input: (Shoot Person2 Person1 ?INSTR6)

Suggesting: ShootToKill1:(ShootToKill Person2 Person1 ?INSTR6)

Suggesting: Murder1:(Murder Person2 Person1)

Found suggested Shoot2:(Shoot Person2 Person1 ?INSTR6)
   matches Shoot1:(Shoot Person2 Person1 ?INSTR6)
  Assuming suggested: Possess2:(Possess Person2 ?INSTR6)
  Assuming suggested: Shot2:(Shot Person1)
  Assuming suggested: BadHealth1:(BadHealth Person1)
  Assuming suggested: BadHealth2:(BadHealth Person1)
  Assuming suggested: State2:(State ?o1)
  Assuming suggested: State1:(State Person1)
  Assuming suggested: GenesisAction1:(GenesisAction Person2)

Input: (Die Person1)

Found suggested Die2:(Die Person1)
   matches Die3:(Die Person1)
  Assuming suggested: Dead1:(Dead Person1)
  Assuming suggested: Die1:(Die Person1)
Determining: ShootToKill1: (ShootToKill Person2 Person1 ?INSTR6)

    Looking for Precondition:(Possess Person2 ?INSTR6)
     Unable to infer (Possess Person2 ?INSTR6)
     Precondition not found,
       Assuming Possess3: (Possess Person2 ?INSTR6)

    Looking for Constraint:(Isa ?INSTR6 Gun)
     Constraint not found,
       Assuming Isa7: (Isa ?INSTR6 Gun)

    Effects:
      Shot2: (Shot Person1)
         Murder2: (Murder Person2 Person1)inferrred from:
            ShootToKill1: (ShootToKill Person2 Person1 ?INSTR6)

Determining: Murder1: (Murder Person2 Person1)
     Isa8: (Isa Person2 Character)inferrred from:
        Isa2: (Isa Person2 Person)

     Found  Isa1: (Isa Person1 Person)
Suggesting: MurderInherit1:(MurderInherit Person2 ?o4-2 Person1)

Found suggested Murder3:(Murder Person2 Person1)
   matches Murder2:(Murder Person2 Person1)
  Assuming suggested: Isa10:(Isa Person2 Character)
  Assuming suggested: Isa11:(Isa Person1 Person)
  Assuming suggested: Dead3:(Dead Person1)
        GenesisAction2: (GenesisAction Person2 . ?x8)inferrred from:
           Shoot2: (Shoot Person2 Person1 ?INSTR6)
        State3: (State Person1)inferrred from:
           Shot2: (Shot Person1)
        State4: (State Person1)inferrred from:
           Shot2: (Shot Person1)

Input: (Gene got $1000000)
```

```
Input: (Isa Money2 Money)
   Add state: Isa14

Input: (Amount Money2 1000000 dollar)
   Add state: Amount2

Input: (Possess Person2 Money2)
   Add state: Possess6
    Unable to infer (Isa Money2 Gun)
    Isa15: (Isa Money2 Valuable)inferred from:
      Isa14: (Isa Money2 Money)

Found suggested Possess5:(Possess Person2 ?o4-2)
  matches Possess6:(Possess Person2 Money2)
  Assuming suggested: Inherit1:(Inherit Person2 Money2 Person1)
  Assuming suggested: Heir1:(Heir Person2 Person1)
  Assuming suggested: Possess4:(Possess Person1 Money2)
  Assuming suggested: Isa13:(Isa Money2 Inanimate)
  Assuming suggested: Isa12:(Isa Money2 Valuable)
  Assuming suggested: Not1:(Not (Possess Person1 Money2))
Determining: MurderInherit1: (MurderInherit Person2 Money2 Person1)

   Looking for Constraint:(Isa Money2 Valuable)
    Found Isa15:(Isa Money2 Valuable)

   Looking for Constraint:(Isa Person1 Person)
    Found Isa1:(Isa Person1 Person)

   Looking for Constraint:(Isa Person2 Character)
    Found Isa8:(Isa Person2 Character)

   Looking for Precondition:(Possess Person1 Money2)
   Add state: LessThan1
    Possess8: (Possess Person1 Money2)inferrred from:
      LessThan1: (LessThan 1000000 order-millions)
      Isa3: (Isa Money1 Money)
      Isa14: (Isa Money2 Money)
      Amount2: (Amount Money2 1000000 dollar)
      Possess1: (Possess Person1 Money1)
      Amount1: (Amount Money1 order-millions dollar)
    ThemeGoalMet1: (ThemeGoalMet Person1 (Possess Person1 Money2)) inferrred from:
      Isa15: (Isa Money2 Valuable)
      Possess8: (Possess Person1 Money2)

   Looking for Precondition:(Heir Person2 Person1)
    Heir2: (Heir Person2 Person1)inferrred from:
      Parent1: (Parent Person1 Person2)

   Effects:
    Dead1: (Dead Person1)
    Not1: (Not (Possess Person1 Money2))
     Not1 deletes Possess8: (Possess Person1 Money2)
    Possess5: (Possess Person2 Money2)
    ThemeGoalMet2: (ThemeGoalMet Person2 (Possess Person2 Money2)) inferrred from:
      Isa15: (Isa Money2 Valuable)
      Possess5: (Possess Person2 Money2)

Thematic goal achieved: ThemeGoalMet2:(ThemeGoalMet Person2 (Possess Person2 Money2))
```

REFERENCES

[Ahn87a] W. Ahn, R. J. Mooney, W. F. Brewer and G. F. DeJong, "Schema Acquisition from One Example: Psychological Evidence for Explanation-Based Learning," *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA, July 1987. (Also appears as Technical Report UILU-ENG-87-2231, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign)

[Ahn87b] W. Ahn, "Schema Acquisition from a Single Example," M.A. Thesis, Department of Psychology, University of Illinois, Urbana, IL, October 1987.

[Anderson79] J. R. Anderson, P. J. Kline and C. M. Beasley, "A General Learning Theory and its Application to Schema Abstraction," in *The Psychology of Learning and Motivation*, G. H. Bower (ed.), Academic Press, New York, NY, 1979, pp. 227-318.

[Anderson82] J. R. Anderson, "Acquisition of Cognitive Skill," *Psychological Review 89*, 4 (1982), pp. 369-406.

[Anderson83a] J. R. Anderson, "Acquisition of Proof Skills in Geometry," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, T. M. Mitchell (ed.), Tioga Publishing Company, Palo Alto, CA , 1983, pp. 191-221.

[Anderson83b] J. R. Anderson, *The Architecture of Cognition*, Harvard University Press, Cambridge, MA, 1983.

[Anderson86] J. R. Anderson, "Knowledge Compilation: The General Learning Mechanism," in *Machine Learning: An Artificial Intelligence Approach, Vol. II*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (ed.), Morgan Kaufman, 1986, pp. 289-309.

[Anderson87a] J. R. Anderson, "Causal Analysis and Inductive Learning," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 288-299.

[Anderson87b] J. R. Anderson and R. Thompson, "Use of Analogy in a Production System Architecture," in *Similarity and Analogical Reasoning*, S. Vosniadou and A. Ortony (ed.), Cambridge University Press, Cambridge, England, 1987.

[Bain86] W. M. Bain, "A Case-Based Reasoning System for Subjective Assessment," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 523-527.

[Bennett86] S. W. Bennett, "Strategy-Based Understanding of Solutions to Mathematical Equations," Working Paper 78, AI Research Group, Coordinated Science Laboratory, University of Illinois, Urbana, IL, March 1986.

[Bennett87] S. W. Bennett, "Approximation in Mathematical Domains," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 239-241.

[Berwick83] R. C. Berwick, "Learning Word Meanings from Examples," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, August 1983, pp. 459-461.

[Bowerman80] M. Bowerman, "The Structure and Origin of Semantic Categories in the Language Learning Child," in *Symbols as Sense: New Approaches to the Analysis of Meaning*, M. L. Foster & S. H. Brandes (ed.), Academic Press, New York, 1980, pp. 277-299.

[Buchanan77] B. G. Buchanan, T. M. Mitchell, R. G. Smith and C. R. Johnson, "Models of Learning Systems," in *Encyclopedia of Computer Science and Technology, Vol. 11*, J. Belzer, A. G. Holzman, & A. Kent (ed.), Marcel Dekker, New York, NY, 1977, pp. 24-51.

[Bundy83] A. Bundy, *The Computer Modelling of Mathematical Reasoning*, Academic Press, New York, NY, 1983.

[Chapman87] D. Chapman, "Planning for Conjunctive Goals," *Artificial Intelligence 32*, 3 (1987), pp. 333-378.

[Charniak77] E. Charniak, "MS. MALAPROP, A Language Comprehension System," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA, August 1977.

[Charniak78] E. Charniak, "With a Spoon in Hand this Must be the Eating Frame," *Theoretical Issues in Natural Language Processing 2*, Urbana, IL, 1978, pp. 187-193.

[Charniak80] E. Charniak, C. Riesbeck and D. McDermott, *Artificial Intelligence Programming*, Lawrence Erlbaum and Associates, Hillsdale, NJ, 1980.

[Charniak82] E. Charniak, "Context Recognition in Language Comprehension," in *Strategies for Natural Language Processing*, W. G. Lehnert and M. H. Ringle (ed.), Lawrence Erlbaum and Associates, Hillsdale, NJ , 1982, pp. 435-454.

[Charniak85] E. Charniak and D. McDermott, *Introduction to Artificial Intelligence*, Addison-Wesley, Reading, MA, 1985.

[Charniak86] E. Charniak, "A Neat Theory of Marker Passing," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 584-588.

[Cheng86] P. Cheng and J. G. Carbonell, "The FERMI System: Inducing Iterative Macro-operators from Experience," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 490-495.

[Chien87a] S. A. Chien, "Simplifications in Temporal Persistence: An Approach to the Intractable Domain Theory Problem in Explanation-Based Learning," M.S. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, August 1987. (Also appears as UILU-ENG-87-2255, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[Chien87b] S. A. Chien, "Extending Explanation-Based Learning: Failure-Driven Schema Refinement," *Proceedings of the Third IEEE Conference on Artificial Intelligence Applications*, Orlando, Florida, February 1987. (Also appears as Technical Report UILU-ENG-87-2203, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[Clocksin84] W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, Springer Verlag, Berlin, 1984.

[Collins80] A. Collins, J. S. Brown and K. M. Larkin, "Inference in Text Understanding," in *Theoretical Issues in Reading Comprehension*, R. Spiro, B. Bruce, W. Brewer (ed.), Lawrence Erlbaum and Associates, Hillsdale, NJ, 1980, pp. 386-410.

[Cullingford78] R. E. Cullingford, "Script Application: Computer Understanding of Newspaper Stories," Technical Report 116, Department of Computer Science, Yale University, New Haven, CT, January 1978.

[Danyluk87] A. P. Danyluk, "The Use of Explanations for Similarity-Based Learning," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 274-276.

[DeJong79] G. F. DeJong, "Skimming Stories in Real Time: An Experiment in Integrated Understanding," Technical Report 158, Ph.D. Thesis, Department of Computer Science, Yale University, New Haven, CT, 1979.

[DeJong81] G. F. DeJong, "Generalizations Based on Explanations," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, B.C., Canada, August 1981, pp. 67-70. (Also appears as Working Paper 30, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[DeJong82a] G. F. DeJong, "Automatic Schema Acquisition in a Natural Language Environment," *Proceedings of the National Conference on Artificial Intelligence*, Pittsburgh, PA, August 1982, pp. 410-413. (Also appears as Working Paper 34, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[DeJong82b] G. F. DeJong, "An Overview of the FRUMP System," in *Strategies for Natural Language Processing*, W. G. Lehnert and M. H. Ringle (ed.), Lawrence Erlbaum and Associates, Hillsdale, NJ , 1982.

[DeJong83] G. F. DeJong, "An Approach to Learning from Observation," *Proceedings of the 1983 International Machine Learning Workshop*, Urbana, IL, June 1983. (Also appears as Working Paper 45, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[DeJong84] G. F. DeJong, A. M. Segre, A. Ram, R. J. Mooney and M. Edel, "A Natural Language Processor that Supports Learning," Working Paper 55, AI Research Group, Coordinated Science Laboratory, University of Illinois, Urbana, IL, March 1984.

[DeJong86a] G. F. DeJong, "Explanation Based Learning," in *Machine Learning: An Artificial Intelligence Approach, Vol. II*, R. S. Michalski, J. G. Carbonell, T. M. Mitchell (ed.), Morgan Kaufmann, Los Altos, CA, 1986.

[DeJong86b] G. F. DeJong and R. J. Mooney, "Explanation-Based Learning: An Alternative View," *Machine Learning 1*, 2 (1986), pp. 145-176.

[de Kleer86] J. de Kleer, "An Assumption-Based Truth Maintenance System," *Artificial Intelligence 28*, (1986), pp. 127-162.

[Dietterich82] T. G. Dietterich, B. London, K. Clarkson and G. Dromney, "Learning and Inductive Inference," in *The Handbook of Artificial Intelligence, Vol. III*, P. R. Cohen & E. A. Feigenbaum (ed.), William Kaufman, Inc., Los Altos, CA, 1982.

[Dietterich83] T. G. Dietterich and R. S. Michalski, "A Comparative Review of Selected Methods for Learning from Examples," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (ed.), Tioga Publishing Company, Palo Alto, CA, 1983, pp. 41-81.

[Dietterich86] T. G. Dietterich, "Learning at the Knowledge Level," *Machine Learning 1*, 3 (1986), pp. 287-316.

[Dijkstra76] E. W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, 1976.

[Doyle79] J. Doyle, "A Truth Maintenance System," *Artificial Intelligence 12*, 3 (1979), pp. 231-272.

[Doyle86] R. Doyle, "Constructing and Refining Causal Explanations from an Inconsistent Domain Theory," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 538-544.

[Dyer83] M. J. Dyer, *In-Depth Understanding*, MIT Press, Cambridge, MA, 1983.

[Ellman87] T. Ellman, "Explanation-Based Learning: A Survey of Programs and Perspectives," Technical Report, Department of Computer Science, Columbia University, New York, NY, 1987.

[Even79] S. Even, *Graph Algorithms*, Computer Science Press, Rockville, MD, 1979.

[Falkenhainer87] B. Falkenhainer, K. D. Forbus and D. Gentner, "The Structure Mapping Engine: Algorithm and Examples," Technical Report UILU-87-1748, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, July 1987.

[Feigenbaum83] E. A. Feigenbaum and P. McCorduck, *The Fith Generation: Artificial Intelligence and Japan's Computer Challenge to the World*, Addison-Wesley, Reading, MA, 1983.

[Fikes72] R. E. Fikes, P. E. Hart and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence 3*, 4 (1972), pp. 251-288.

[Flann86] N. S. Flann and T. G. Dietterich, "Selecting Appropriate Representations for Learning from Examples," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 460-466.

[Gangly87] B. Gangly, "The Devolving Science of Machine Learning," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 398-401.

[Garey79] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[Gerrold73] D. Gerrold, *The World of Star Trek*, Ballantine, New York, NY, 1973.

[Gick83] M. L. Gick and K. L. Holyoak, "Schema Induction and Analogical Transfer," *Cognitive Psychology 15*, (1983), pp. 1-38.

[Gopnik86] A. Gopnik and A. N. Meltzoff, "Relations between Semantic and Cognitive Development in the One-Word Stage: The Specificity Hypothesis," *Child Development 57*, (1986), pp. 1040-1053.

[Granger77] R. H. Granger, "FOUL-UP: A Program that Figures Out Meanings of Words from Context," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA, August 1977, pp. 172-178.

[Granger80] R. H. Granger, "Adaptive Understanding: Correcting Erroneous Inferences," Technical Report 171, Yale University, Department of Computer Science, New Haven, Conn., January 1980.

[Gupta87] A. Gupta, "Explanation-Based Failure Recovery," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 606-610.

[Hall85] R. P. Hall and D. F. Kibler, "Differing Methodological Perspectives in Artificial Intelligence Research," *Artificial Intelligence Magazine 6*, 3 (1985), pp. 166-178.

[Hirsh87] H. Hirsh, "Explanation-Based Generalization in a Logic-Programming Environment," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987.

[Hirst87] G. Hirst, *Semantic Interpretation and the Resolution of Ambiguity*, Cambridge University Press, Cambridge, England, 1987.

[Iba85] G. A. Iba, "Learning by Discovering Macros in Puzzle Solving," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 640-642.

[Jurgenson75] R. C. Jurgenson, A. J. Donnelly, J. E. Maier and G. R. Rising, *Geometry*, Houghton Mifflin, Boston, 1975.

[Kass86] A. Kass, "Modifying Explanations to Understand Stories," *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA, August 1986, pp. 691-696.

[Kautz86] H. A. Kautz and J. F. Allen, "Generalized Plan Recognition," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 32-37.

[Kedar-Cabelli87a] S. T. Kedar-Cabelli and L. T. McCarty, "Explanation-Based Generalization as Resolution Theorem Proving," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 383-389.

[Kedar-Cabelli87b] S. T. Kedar-Cabelli, "Formulating Concepts According To Purpose," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 477-481.

[Keller87a] R. M. Keller, "The Role of Explicit Contextual Knowledge in Learning Concepts to Improve Performance," Ph.D. Thesis, Department of Computer Science, Rutgers University, New Brunswick, NJ, January 1987. (Also appears as Machine Learning Technical Report #7, Laboratory for Computer Science Research, Rutgers University)

[Keller87b] R. M. Keller, "Defining Operationality for Explanation-Based Learning," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 482-487.

[Kernighan79] B. W. Kernighan, "PIC -- A Graphics Language for Typesetting," in *UNIX Programmer's Manual, vol. 2*, Bell Laboratories, Murray Hill, N.J., 1979.

[Kolodner84] J. L. Kolodner, *Retrieval and Organization Strategies in Conceptual Memory*, Lawrence Erlbaum and Associates, Hillsdale, NJ, 1984.

[Kolodner87] J. L. Kolodner, "Extending Problem Solver Capabilities Through Case-Based Inference," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 167-178.

[Korf85] R. E. Korf, "Macro-Operators: A Weak Method for Learning," *Artificial Intelligence 26*, (1985), pp. 35-77.

[Kowalski79] R. Kowalski, *Logic for Problem Solving*, Elsevier North Holland, New York, NY, 1979.

[Kuczaj82] S. A. Kuczaj, "Acquisition of Word Meaning in the Context of the Development of the Semantic System," in *Verbal Processes in Children: Progress in Cognitive Developmental Research*, C. J. Brainerd & M. Pressley (ed.), Springer-Verlag, New York, 1982, pp. 95-123.

[Kuhn70] T. S. Kuhn, *The Structure of Scientific Revolutions, 2nd ed.*, University of Chicago Press, Chicago, IL, 1970.

[Laird84] J. Laird, P. Rosenbloom and A. Newell, "Towards Chunking as a General Learning Mechanism," *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, August 1984, pp. 188-192.

[Laird86a] J. Laird, P. Rosenbloom and A. Newell, "Chunking in Soar: The Anatomy of a General Learning Mechanism," *Machine Learning 1*, 1 (1986), pp. 11-46.

[Laird86b] J. E. Laird, P. S. Rosenbloom and A. Newell, *Universal Subgoaling and Chunking: The Automatic Generation and Learning of Goal Hierarchies*, Kluwer Academic Publishers, Norwell, MA, 1986.

[Langley81] P. Langley, "Data-Driven Discovery of Physical Laws," *Cognitive Science 5*, 1 (1981), pp. 31-54.

[Langley86] P. Langley, "Editorial: The Terminology of Machine Learning," *Machine Learning 1*, 2 (1986), pp. 141-144.

[Leake86] D. B. Leake and C. C. Owens, "Organizing Memory for Explanation," *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA, August 1986, pp. 710-715.

[Lebowitz80] M. Lebowitz, "Generalization and Memory in an Integrated Understanding System," Technical Report 186, Ph.D Thesis, Department of Computer Science, Yale University, New Haven, CT, 1980.

[Lebowitz86] M. Lebowitz, "Integrated Learning: Controlling Explanation," *Cognitive Science 10*, 2 (1986), pp. 219-240.

[Lehnert78] W. Lehnert, *The Process of Question Answering*, Lawrence Erlbaum and Associates, Hillsdale, NJ, 1978.

[Lehnert82] W. G. Lehnert, "Plot Units: A Narrative Summarization Strategy," in *Strategies for Natural Language Processing*, W. G. Lehnert and M. H. Ringle (ed.), Lawrence Erlbaum and Associates, Hillsdale, NJ, 1982, pp. 375-414.

[Lehnert84] W. G. Lehnert, "Paradigmatic Issues in Cognitive Science," in *Method and Tactics in Cognitive Science*, W. Kintsch, J. R. Miller & P. G. Polson (ed.), Lawrence Erlbaum and Associates, Hillsdale, NJ, 1984, pp. 21-50.

[Lesk79] M. E. Lesk, "TBL -- A Program that Formats Tables," in *UNIX Programmer's Manual, vol. 2*, Bell Laboratories, Murray Hill, N.J., 1979, pp. Section 10..

[Lloyd84] J. W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, New York, NY, 1984.

[Lytinen84] S. L. Lytinen, "Frame Selection in Parsing," *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, August 1984.

[Mahadevan85] S. Mahadevan, "Verification-Based Learning: A Generalization Strategy for Inferring Problem-Reduction Methods," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 616-623.

[Manna74] Z. Manna, *Mathematical Theory of Computation*, McGraw-Hill, New York, NY, 1974.

[Marcus74] M. Marcus, "Wait-and-See Strategies for Parsing Natural Language," Working Paper 75, MIT AI Lab, Cambridge, MA, August 1974.

[Marcus80] M. P. Marcus, *A Theory of Syntactic Recognition for Natural Language*, MIT Press, Cambridge, MA, 1980.

[McAllester82] D. A. McAllester, "Reasoning Utility Package User's Manual, Version One," Memo 667, MIT AI Lab, Cambridge, MA, April 1982.

[Medin87a] D. L. Medin, W. D. Wattenmaker and S. E. Hampson, "Family Resemblance, Conceptual Cohesiveness, and Category Construction," *Cognitive Psychology 19*, (1987), pp. 242-279.

[Medin87b] D. L. Medin, W. D. Wattenmaker and R. S. Michalski, "Constraints and Preferences in Inductive Learning: An Experimental Study of Human and Machine Performance," *Cognitive Science 11*, 3 (1987), pp. 299-239.

[Meehan76] J. Meehan, "The Metanovel: Writing Stories by Computer," Technical Report 74, Ph.D. Thesis, Yale University, New Haven, CT, 1976.

[Michalski86] R. S. Michalski, "Understanding the Nature of Learning: Issues and Research Directions," in *Machine Learning: An Artificial Intelligence Approach, Vol. II*, R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (ed.), Morgan Kaufman Inc., Los

Altos, CA, 1986, pp. 3-26.

[Minton84] S. N. Minton, "Constraint-Based Generalization: Learning Game-Playing Plans from Single Examples," *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, August 1984, pp. 251-254.

[Minton85] S. N. Minton, "Selectively Generalizing Plans for Problem-Solving," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 596-599.

[Minton87a] S. Minton, J. G. Carbonell, O. Etzioni, C. A. Knoblock and D. R. Kuokka, "Acquiring Effective Search Control Rules: Explanation-Based Learning in the PRODIGY System," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 122-133.

[Minton87b] S. Minton and J. G. Carbonell, "Strategies for Learning Search Control Rules: An Explanation-based Approach," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 228-235.

[Mitchell82] T. M. Mitchell, "Generalization as Search," *Artificial Intelligence 18*, 2 (1982), pp. 203-226.

[Mitchell83] T. M. Mitchell, "Learning and Problem Solving," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, August 1983, pp. 1139-1151.

[Mitchell84] T. M. Mitchell, "Toward Combining Empirical and Analytic Methods for Learning Heuristics," in *Human and Artificial Intelligence*, A. Elithorn & R. Banerji (ed.), North-Holland, Amsterdam, 1984.

[Mitchell85] T. M. Mitchell, S. Mahadevan and L. I. Steinberg, "LEAP: A Learning Apprentice for VLSI Design," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 573-580.

[Mitchell86] T. M. Mitchell, R. Keller and S. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," *Machine Learning 1*, 1 (January 1986), pp. 47-80.

[Mogensen87] B. N. Mogensen, "Goal-Oriented Conceptual Clustering: The Classifying Attribute Approach," M.S. Thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL, 1987.

[Mooney85a] R. J. Mooney and G. F. DeJong, "Learning Schemata for Natural Language Processing," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 681-687.

[Mooney85b] R. J. Mooney, "Generalizing Explanations of Narratives into Schemata," M.S. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, May 1985. (Also appears as Technical Report T-159, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[Mooney86a] R. J. Mooney, "Generalizing Explanations of Narratives into Schemata," in *Machine Learning: A Guide To Current Research*, T. M. Mitchell, J. G. Carbonell and R. S. Michalski (ed.), Kluwer Academic Publishers, Hingham, MA, 1986, pp. 207-212.

[Mooney86b] R. J. Mooney and S. W. Bennett, "A Domain Independent Explanation-Based Generalizer," Technical Report UILU-ENG-86-2216, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, June, 1986.

[Mooney86c] R. J. Mooney and S. W. Bennett, "A Domain Independent Explanation-Based Generalizer," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 551-555. (A longer updated version appears as Technical Report UILU-ENG-86-2216, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign)

[Mooney87a] R. J. Mooney, "Integrated Learning of Words and their Underlying Concepts," *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA, July 1987. (Also appears as Technical Report UILU-ENG-87-2229, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign)

[Mooney87b] R. J. Mooney and G. F. DeJong, "Learning Indices for Conceptual Information Retrieval," Technical Report UILU-ENG-87-2230, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, May 1987.

[Murphy85] G. L. Murphy and D. L. Medin, "The Role of Theories in Conceptual Coherence," *Psychological Review 92*, 3 (July 1985), pp. 289-316.

[Newell73] A. Newell, "Artificial Intelligence and the Concept of Mind," in *Computer Models of Thought and Language*, R. C. Schank & K. M. Colby (ed.), W. H. Freeman and Company, San Francisco, CA, 1973, pp. 1-60.

[Nilsson80] N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing Company, Palo Alto, CA, 1980.

[Norvig83] P. Norvig, "Frame Activated Inferences in a Story Understanding Program," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, August 1983, pp. 624-626.

[O'Rorke83] P. V. O'Rorke, "Reasons for Beliefs in Understanding: Applications of Non-Monotonic Dependencies to Story Processing," *Proceedings of the National Conference on Artificial Intelligence*, Washington, D.C., August 1983. (Also appears as Working Paper 43, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[O'Rorke84] P. V. O'Rorke, "Generalization for Explanation-based Schema Acquisition," *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, August 1984, pp. 260-263.

[O'Rorke87a] P. V. O'Rorke, "LT Revisited: Experimental Results of Applying Explanation-Based Learning to the Logic of Principia Mathematica," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 148-159.

[O'Rorke87b] P. V. O'Rorke, "Explanation-Based Learning Via Constraint Posting and Propagation," Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, January 1987. (Also appears as UILU-ENG-87-2239, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[Orejel-Opisso84] J. Orejel-Opisso, "Story Understanding with WATSON, a Computer Program Modelling Natural Language Inferences using Non-Monotonic Dependencies," M.S. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, August 1984. (Also appears as Technical Report T-146, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[Paterson78] M. S. Paterson and M. N. Wegman, "Linear Unification," *Journal of Computer and System Sciences 16*, (1978), pp. 158-167.

[Pazzani87] M. J. Pazzani, "Inducing Causal and Social Theories: A Prerequisite for Explanation-Based Learning," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 230-241.

[Porter85] B. Porter and D. Kibler, "A Comparison of Analytic and Experimental Goal Regression for Machine Learning," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 555-559.

[Porter86] B. W. Porter and D. F. Kibler, "Experimental Goal Regression: A Method for Learning Problem-solving," *Machine Learning 1*, 3 (1986), pp. 249-286.

[Posner68] M. J. Posner and S. W. Keele, "On the Genesis of Abstract Ideas," *Journal of Experimental Psychology 77*, 3 (July 1968), pp. 353-363.

[Prieditis86] A. E. Prieditis, "Discovery of Algorithms from Weak Methods," *Proceedings of the International Meeting on Advances in Learning*, Les Arcs, Switzerland, 1986, pp. 37-52.

[Prieditis87] A. E. Prieditis and J. Mostow, "PROLEARN: Towards a Prolog Interpreter that Learns," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 494-498.

[Rajamoney85] S. Rajamoney, G. F. DeJong and B. Faltings, "Towards a Model of Conceptual Knowledge Acquisition through Directed Experimentation," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985. (Also appears as Working Paper 68, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[Rajamoney87] S. Rajamoney and G. DeJong, "The Classification, Detection and Handling of Imperfect Theory Problems," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 205-207.

[Reingold77] E. M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[Robinson65] J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *Journal of the Association for Computing Machinery 12*, 1 (1965), pp. 23-41.

[Rosenbloom86] P. Rosenbloom and J. Laird, "Mapping Explanation-Based Generalization into Soar," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 561-567.

[Rumelhart78] D. E. Rumelhart and D. A. Norman, "Accretion, Tuning, and Restructuring: Three Modes of Learning," in *Semantic Factors in Cognition*, J. W. Cotton & R. L. Klatzky (ed.), Lawrence Erlbaum and Associates, Hillsdale, NJ, 1978.

[Russell85] S. Russell, "The Compleat Guide to MRS," Technical Report KSL 85-12, Computer Science Department, Stanford University, June 1985.

[Sacerdoti74] E. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence 5*, (1974), pp. 115-135.

[Sacerdoti77] E. Sacerdoti, *A Structure for Plans and Behavior*, American Elsevier, New York, 1977.

[Schank75] R. C. Schank, *Conceptual Information Processing*, North-Holland/American Elsevier, Amsterdam, 1975.

[Schank77] R. C. Schank and R. P. Abelson, *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures*, Lawrence Erlbaum and Associates, Hillsdale, NJ, 1977.

[Schank81a] R. C. Schank, J. L. Kolodner and G. F. DeJong, "Conceptual Information Retrieval," in *Information Retrieval Research*, R. N. Oddy, S. E. Robertson, C. J. van Rijsbergen, and P. W. Williams (ed.), Butterworths, London, 1981.

[Schank81b] R. C. Schank and C. Riesbeck, *Inside Computer Understanding*, Lawrence Erlbaum and Associates, Hillsdale, NJ, 1981.

[Schank82] R. C. Schank, *Dynamic Memory*, Cambridge University Press, Cambridge, England, 1982.

[Schank86a] R. C. Schank, *Explanation Patterns: Understanding Mechanically and Creatively*, Lawrence Erlbaum and Associates, Hillsdale, NJ, 1986.

[Schank86b] R. C. Schank, G. C. Collins and L. E. Hunter, "Transcending Inductive Category Formation in Learning," *Behavioral and Brain Sciences 9*, (1986), pp. 639-686.

[Schmidt78] C. F. Schmidt, N. S. Sridharan and J. L. Goodson, "The Plan Recognition Problem: An Intersection of Psychology and Artificial Intelligence," *Artificial Intelligence 11*, (1978), pp. 45-83.

[Sedgewick83] R. Sedgewick, *Algorithms*, Addison-Wesley, Reading, MA, 1983.

[Segre85] A. M. Segre and G. F. DeJong, "Explanation Based Manipulator Learning: Acquisition of Planning Ability Through Observation," *Proceedings of the IEEE International Conference on Robotics and Automation*, St. Louis, MO, March 1985, pp. 555-560. (Also appears as Working Paper 62, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[Segre87a] A. M. Segre, "Explanation-Based Learning of Generalized Robot Assembly Tasks," Ph.D. Thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL, January 1987. (Also appears as UILU-ENG-87-2208, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[Segre87b] A. M. Segre, "On the Operationality/Generality Trade-off in Explanation-Based Learning," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987.

[Selfridge82] M. Selfridge, "Inference and Learning in a Computer Model of the Development of Langague Comprehension in a Young Child," in *Strategies for Natural Language Processing*, W. G. Lehnert and M. H. Ringle (ed.), Lawrence Erlbaum and Associates, 1982, pp. 299-326.

[Shavlik85] J. W. Shavlik, "Learning about Momentum Conservation," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 667-669. (Also appears as Working Paper 66, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[Shavlik87a] J. W. Shavlik and G. F. DeJong, "BAGGER: An EBL System that Extends and Generalizes Explanations," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 516-520.

[Shavlik87b] J. W. Shavlik, G. F. DeJong and B. H. Ross, "Acquiring Special Case Schemata in Explanation-Based Learning," *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA, July 1987, pp. 851-860.

[Shavlik88] J. W. Shavlik, "Generalizing the Structure of Explanations in Explanation-Based Learning ," Ph.D. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, January 1988.

[Silver83] B. Silver, "Learning Equation Solving Methods from Worked Examples," *Proceedings of the 1983 International Machine Learning Workshop*, Urbana, IL, June 1983, pp. 99-104.

[Silver86] B. Silver, "Precondition Analysis: Learning Control Information," in *Machine Learning: An Artificial Intelligence Approach, Vol. II*, R. S. Michalski, J. G.

Carbonell and T. M. Mitchell (ed.), Morgan Kaufmann, Los Altos, CA, 1986, pp. 647-670.

[Simon83] H. A. Simon, "Why Should Machines Learn?," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (ed.), Tioga Publishing Co., Palo Alto, CA., 1983, pp. 25-37.

[Spencer86] R. M. Spencer and R. W. Weisberg, "Context-Dependent Effects on Analogical Transfer," *Memory and Cognition 14, 5* (1986), pp. 442-449.

[Stepp86] R. E. Stepp and R. S. Michalski, "Conceptual Clustering of Structured Objects: A Goal-Oriented Approach," *Artificial Intelligence 28*, 1 (Feb 1986), pp. 43-69.

[Sussman73] G. J. Sussman, "A Computational Model of Skill Acquisition," Technical Report 297, MIT AI Lab, Cambridge, MA, 1973.

[Tadepalli86] P. V. Tadepalli, "Learning in Intractable Domains," in *Machine Learning: A Guide To Current Research*, T. M. Mitchell, J. G. Carbonell and R. S. Michalski (ed.), Kluwer Academic Publishers, Hingham, MA, 1986, pp. 337-342.

[Tate76] A. Tate, "Project Planning Using a Hierarchic Nonlinear Planner," Technical Report 25, Department of Artificial Intelligence Research, University of Edinburgh, Edinburgh, Scotland, 1976.

[Teitelman83] W. Teitelman, *Interlisp Reference Manual*, Xerox PARC, Palo Alto, CA, 1983.

[Thorndyke79] P. W. Thorndyke and B. Hayes-Roth, "The Use of Schemata in the Acquisition and Transfer of Knowledge," *Cognitive Psychology 11*, (1979), pp. 82-106.

[Utgoff86] P. E. Utgoff, "Shift of Bias for Inductive Concept Learning," in *Machine Learning: An Artificial Intelligence Approach, Vol. II*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (ed.), Morgan Kaufman, 1986, pp. 107-148.

[Waldinger77] R. Waldinger, "Achieving Several Goals Simultaneously," in *Machine Intelligenge 8*, E. Elcock and D. Michie (ed.), Ellis Horwood Limited, London, 1977.

[Waltz78] D. L. Waltz, "An English Language Question Answering System for a Large Relational Data Base," *Communications of the Association for Computing Machinery 21*, 7 (July 1978), pp. 526-539.

[Waltz84] D. L. Waltz and J. B. Pollack, "Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation," *Cognitive Science*, 1984. (A version of this paper appears as Working Paper 48, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.)

[Whitehead13] A. N. Whitehead and B. Russell, *Principia Mathematica*, Cambridge University Press, Cambridge, England, 1913.

[Wilensky78] R. W. Wilensky, "Understanding Goal-Based Stories," Technical Report 140, Ph.D. Thesis, Department of Computer Science, Yale University, New Haven, CT, September 1978.

[Wilensky80] R. Wilensky and Y. Arens, "PHRAN: A Knowledge-Based Approach to Natural Language Analysis," Electronic Research Laboratory Memorandum No. UCB/ERL/M80/34, University of California, Berkely, CA, 1980.

[Wilensky82] R. Wilensky, "Points: A Theory of the Structure of Stories in Memory," in *Strategies for Natural Language Processing*, W. G. Lehnert and M. H. Ringle (ed.), Lawrence Erlbaum and Associates, Hillsdale, NJ, 1982, pp. 345-73.

[Wilensky83] R. W. Wilensky, *Planning and Understanding: A Computational Approach to Human Reasoning*, Addison-Wesley, Reading, MA, 1983.

[Wilkins86] D. C. Wilkins, W. J. Clancey and B. G. Buchanan, "ODYSSEUS: A Learning Apprentice," in *Machine Learning: A Guide To Current Research*, T. M. Mitchell, J. G. Carbonell and R. S. Michalski (ed.), Kluwer Academic Publishers, Hingham, MA, 1986, pp. 369-374.

[Wilks75] Y. A. Wilks, "Primitives and Words," in *Theoretical Issues in Natural Language Processing*, R. C. Schank and B. Nash-Webber (ed.), Association for Computational Linguistics, Arlington, VA, 1975, pp. 38-41.

[Winston83] P. H. Winston, T. O. Binford, B. Katz and M. Lowry, "Learning Physical Descriptions from Functional Definitions, Examples, and Precedents," *Proceedings of the National Conference on Artificial Intelligence*, Washington, D.C., August 1983, pp. 433-439.

[Xerox86] Xerox, *The Lisp Library Packages Manual*, Xerox Corporation, Pasedena, CA., 1986.

VITA

Raymond Joseph Mooney was born on September 27, 1961 in St. Louis, Missouri. He attended the University of Illinois at Urbana-Champaign where he received a B.S. degree in Computer Engineering in 1983. Later that year he entered the graduate program in Computer Science and subsequently became a research assistant for Professor Gerald DeJong in the Coordinated Science Laboratory. He received the Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in January 1988.

His interests include machine learning, natural language understanding, and cognitive science. He has co-authored an article on explanation-based learning for the *Machine Learning* journal and has presented papers at several scientific conferences on artificial intelligence and cognitive science.

He is currently an Assistant Professor in the Department of Computer Sciences at the University of Texas at Austin.