Goal Analysis:
Plan Recognition in Dialogue Systems

by
James Mayfield

Abstract

Plan recognition is an important aspect of dialogue processing, because it provides access to the speaker's intentions. *Goal analysis* is plan recognition that focuses on inferring the speaker's goals. This dissertation describes a goal analyzer called **PAGAN** that is part of the UNIX Consultant project.

This dissertation makes four main contributions. First, it provides a theory of how the merits of an explanation of an utterance may be judged. There are three criteria for making such judgments: the *applicability* of the explanation to the system's needs; the *grounding* of the explanation in what is already known of the speaker and of the dialogue; and the *completeness* of the explanation's coverage of the speaker's goals.

The second main contribution of this work is the introduction of a representation for plans and goals that solves two problems inherent in many approaches to plan representation. First, there is usually an inadequate distinction between the effects of an action and *the* effect that the action was intended to produce. Secondly, the traditional notion of a precondition fails to distinguish between heuristic planning knowledge and knowledge of the defining properties of actions. The solution to these problems, called a *planfor*, is a relation between a type of goal and a sequence of hypothetical actions that constitutes a possible method of achieving a goal of that type.

The third contribution of this work is a detailed analysis of ambiguity. The sources of ambiguity include ambiguities at the sentence and utterance levels, ambiguity arising from the existence of multiple explanatory plan schemas, ambiguity arising when it is possible to place a mentioned concept into more than one competing category, and ambiguity arising when it is unclear which concept is the topic of a particular relation (such as a question). This dissertation suggests a unified approach to handling these types of ambiguities.

The fourth contribution of this work is the introduction of a method for determining how much processing of an utterance should be done. It is based on an assessment of the completeness of the explanation, and on an assessment of the practicability of continuing the processing.

Acknowledgements*

This dissertation would not have been possible without the guidance of my advisor, Robert Wilensky. He conceived the UNIX Consultant and provided the foundation for the work I present here. My other readers, Lotfi Zadeh and Paul Kay, were patient and always gracious. I also want to thank the other members of my committee, Richard Karp and Richard Alterman, for giving me a vote of confidence in the fledgling stages of this work.

The Berkeley Artificial Intelligence Research Lab is an amazing source of enlightenment, diversion (principal investigator: Dan Jurafsky), and dubious humor (principal investigator: Michael Braverman). Jim Martin and Marc 'Diet Coke' Luria, whose tenure at BAIR closely paralleled my own, provided among other things reams of good advice about what to do and what not to do in constructing a dissertation. They were especially skilled at deciphering cryptic scrawls on drafts of this and other theses. Other members of BAIR who assisted me in the production of this dissertation are (in reverse alphabetical order by last letter) Dan 'Boychick' Jurafsky, Charley Cox, Dekai Wu, Lisa 'Strange Greeting' Rau, Anthony Albert, Yigal Arens, Paul Jacobs, Margaret Butler, Terry 'Dr. Fell' Regier, Eric Karlson, David Chin, Michael 'Moooo' Braverman, Joe Faletti, Peter Norvig, and Nigel 'If I had a gun' Ward.

I doubt that much of anything would get done at BAIR without the helping hand of Sharon 'What are you going to do, run me over?' Tague. Her special mixture of optimism, provocative conversation, and heterodoxy made life in Evans hall seem less like life in Evans hall than it probably was. Thank you, Sharon.

Five long-time roommates, Ross Ihaka, Richard Berger, Rene Peralta, Steve Lewis, and McGannahan Skjellyfetti, gave me both the pleasure of their company during my stay in Berkeley and the layperson's perspective on my work. Steve in particular demonstrated by example the value of a steady, consistent dedication to one's work; for this I am deeply indebted to him.

Two people from my pre-graduate school days steered me toward the study of natural language processing. Eric Roberts imparted to me both a love of and a solid foundation in computer science. He also taught me quite a bit about juggling, Whovianism, social responsibility, and a host of other topics. Michael Creech introduced me to natural language processing; I ignored his warnings and got caught up in his enthusiasm.

Sponsored by the Defense Advanced Research Projects Agency (DoD) monitored by Space and Naval Warfare Systems Command under Contract N00039-88-C-0292; and through the Office of Naval Research. under Contract N00014-80-C-0732.

For the past year and some months, the Computer Science Department at the University of Maryland, Baltimore County has provided me with the resources I have needed to complete this dissertation. Samuel Lomonaco, the department Chair, has not only assisted me in many ways on this project, but has also put together a superb environment for research in computer science.

My parents, Clifton and Jane Mayfield, have always seemed to supply just the right mixture encouragement and noninterference. Their commitment to education and their unflagging personal support have helped me through many tough times (more than one associated with this dissertation), and have sweetened the not-so-tough ones. I also want to thank my siblings, Clif, Beth, and Marcia, who helped me with this project more than they realize.

My constant companion and friend almost from the start of this project has been Karen Clark. Although she has been disturbingly good at distracting me from my writing, she has also coaxed me along at crucial points. She gave me a rhetorician's view of my subject matter and tried her best (often unsuccessfully) to hold my butchery of the English language in check. I look forward to her distractions long after I've forgotten that I wrote this dissertation.

Lastly, I'd like to thank Jim Martin for writing Marc Luria's acknowledgements.

Contents

Chapter 1

Introduction

1.1. Goal Analysis

Consider the following conversations between a novice computer user who is trying to find out how to delete one of her[1] files, and two consultants. The user's dialogue with a competent consultant might go something like this:

User: Can you tell me how to delete a file?
Consultant: Use rm.

In contrast, consider the same dialogue held with an incompetent consultant:

User: Can you tell me how to delete a file?
[1] **Consultant**: I sure can.
User: What is it?
[2] **Consultant**: What is what?
User: What is a way to delete a file?
[3] **Consultant**: It is a method which, when executed, causes an existing file to become non-existent.
User: No, tell me how I can delete a file.
[4] **Consultant**: By deleting the file called ~/.login.
User: But I want to delete the file transcript.34.
[5] **Consultant**: Thank you for sharing your desires.
User: Forget it! Goodbye.
[6] **Consultant**: Transcript of session saved in file transcript.35. Goodbye.

[1] Throughout this thesis, the user described in the examples is female. The selection of gender was determined by a flip of the coin.

The user had less success in this dialogue because the consultant was consistently unable to correctly infer the *goals* she was pursuing. For example, in the first response, the consultant has interpreted as a direct question what is intended by the user to be an indirect request to be told how to delete a file. In the third response, the consultant has treated a question that asks for a specific version of a specified concept, in this case file deletion, as a request for a description of what it means to be a method for file deletion. In the fourth response, the consultant has misunderstood which component of the question should be further specified. The consultant is only successful in addressing the user's goals in its very last utterance.

This thesis is an approach to the problem of goal analysis. Briefly stated, goal analysis is the process of determining what goals a speaker is addressing in making an utterance. Goal analysis then is a crucial component of a consultant system, and in fact of any dialogue system.

A *goal analyzer* is a system that performs goal analysis. The purpose of a goal analyzer is threefold:

1. To infer part of the 'meaning' of an utterance
2. To suggest behavior
3. To simulate users' understanding

The first purpose of a goal analyzer is to continue the process, begun by the parser and semantic interpreter, of inferring the 'meaning' of an utterance. I construe the word 'meaning' here to include both speaker's meaning (that is, the meaning the speaker *intends* to convey in making the utterance), and any additional facts the hearer may be able to glean from the utterance, regardless of whether the speaker intended to convey them. Together, these components of the 'meaning' of an utterance form an *explanation* of that utterance. For example, an explanation of the utterance:

User: How do I delete a file?

might include (among others) the following facts:

1. The user wants to know how to delete a file.
2. The user wants UC to tell her how to delete a file.
3. The user wants to delete a file.
4. The user wants to free up disk space.

Secondly, a goal analyzer provides the system that uses it with suggestions for future behavior. Such suggestions are of two types. One type of suggested behavior is the continuation of a plan initiated by the user, as when the system answers a question that the user has asked. The other type of suggested behavior is the planning and execution of a new method for achieving an inferred goal. For example, if the user asks a question of the system, the system may indicate some other way that the user can find the answer to the question. Notice that these two types of responses are only suggestions, in that the system is free to pursue some other tack; the goal analyzer does not dictate the

operation of the system as a whole. For example, the system could decide to ignore the user's goals altogether, and instead pursue its own agenda in the dialogue.

The third purpose of a goal analyzer is to simulate the user's understanding of the system's utterances. In addition to analyzing the user's utterances, the system can invoke the goal analyzer on its own utterances. Doing so can give the system valuable information about how the user is likely to interpret the system's utterances. For example, performing goal analysis on its own utterance might alert the system to a potential misinterpretation of that utterance.

This dissertation describes a goal analyzer called **PAGAN** (Plan And Goal ANalyzer) that serves these purposes. **PAGAN** is both a theoretical approach to goal analysis and an implementation of this approach. However, it will always be obvious from context which of these meanings of '**PAGAN**' is intended when I use the term.

1.2. Background

There are two important systems that form the background against which the theories described herein were developed. The first is the UNIX Consultant [Wilensky et al. 1986, 1989], the system in which the implementation of **PAGAN** is designed to operate; the second is **KODIAK** [Wilensky, 1987a], the representation language that **PAGAN** uses to represent both the user's utterances and its knowledge about dialogue and about the world. The UNIX Consultant and **KODIAK** are described in detail in the following sections.

1.2.1. The UNIX Consultant

The work presented herein was done in conjunction with the UNIX Consultant (UC) project at the University of California, Berkeley [Wilensky et al. 1986, 1989]. UC is designed to simulate a computer consultant. Its purpose is to aid novice UNIX users by answering questions and engaging in limited dialogue. A picture of UC's overall architecture is shown in Figure 1.1.

UC's processing of a user's utterance is broken down into several stages. First, the utterance is parsed, and a representation of the utterance expressed in **KODIAK** (see below) is produced. Simple categorization inferences are made on this representation by a mechanism called a *concretion mechanism* (See Chapter 5 for a description of concretion inferences), and the result is handed as input to **PAGAN**. **PAGAN** determines what the user's goals were in making the utterance, and hands the plan and goal structure it produces to the **UCEgo** component [Chin 1988]. **UCEgo** is the portion of the system that serves as an agent; it decides which goals UC should pursue. If necessary, the **KIP** planner [Luria 1988] is invoked to solve a domain goal. Finally, UC's response is selected by the **UCExpress** mechanism and generated in English by the **UCGen** generator.

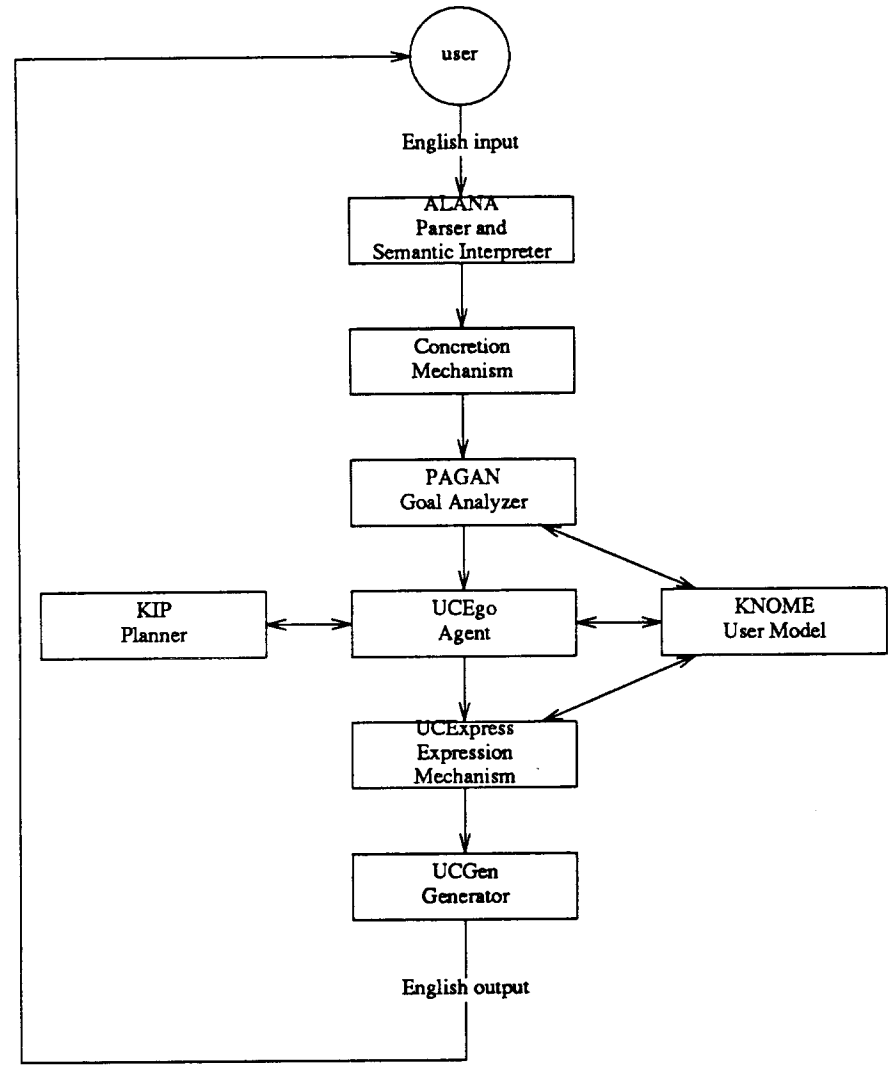

Figure 1.1. Organization of the UNIX Consultant

Running alongside these components is the **KNOME** user model [Chin 1988]. There has been some controversy in the literature over whether dialogue models and user models can or should be separate entities (see Kobsa and Wahlster [1987] for a discussion of these issues). I claim that it is important to separate the two for the purposes of disambiguation (as described in Chapter 7). Thus, I break up the task of inferring and managing knowledge about dialogue participants into *user modeling* and *goal analysis*. User modeling encompasses the collection and maintenance of static knowledge of the user, that is, knowledge that is not likely to change drastically from one moment to the next. User modeling under my definition then is concerned with keeping track of what the speaker does and does not know. While individual facts may be added to or deleted from the user model over the course of a dialogue, in aggregate the model remains relatively stable.

Goal analysis on the other hand is concerned with the more dynamic types of knowledge about the user: what goals the user holds, what plans she is using to try to achieve those goals, and what kinds of responses she expects from her audience. Of course, in overview, the user's high-level domain goals may change very little over the course of a dialogue. However, the user's dialogue-level goals will change rapidly as the dialogue moves from one domain topic to another.

The following trace shows several sessions with UC. It demonstrates the context in which PAGAN operates. The trace and comments are taken directly from Wilensky et al. [1989].

% UC
Welcome to UC (Unix Consultant) version 3.23
To a UC '#' prompt, please type in your questions
about the Unix file system in English.
To leave, just type a '^D' or '(exit)'.

Hi.
How can I help you?

Is rn used to rename files?
No, rn is used to read news.
To rename a file to another file, use mv.
For example, to rename the file named foo to the file named foo1,
type 'mv foo foo1'.

In this example, UC *hypothesized that the user asked the question in order to get an answer to some underlying question. In particular,* PAGAN *hypothesized possible goals of wanting to know how to rename a file and wanting to know what 'rn' does. Since the answer to the direct question is negative,* UCEgo *realizes that the underlying goals are not addressed by this answer, and, therefore, attempts to answer them. This results in the extended answer.*

How can I crash the system?
I'm sorry, I cannot tell you.

In this example, UC *detected a conflict between its goal of helping the user and that of preserving the integrity of the system. It resolved this conflict in favor of the latter goals. However, since it could not accommodate the user's request, it attempts to be polite by apologizing to the user.*

What does ls -v do?
Ls does not have a -v option.

Here the user asks about a non-existent command option. Actually, the UNIX 'ls' command ignores options that it does not recognize, so the consultant could have answered truthfully by saying that 'ls -v' causes one's directory to be listed. It did not, because **PAGAN** *interpreted the request as a question about the function associated with the command.* **UCEgo** *reasons that, since (1) it doesn't know about such an option, and (2) it knows all the options to all simple commands, and (3) 'ls' is a simple command, the user must have a misconception. It then generates the goals of correcting the misconception, which results in the answer shown. Note that* **UC** *uses an open world hypothesis together with metaknowledge to determine whether it knows some facts about UNIX.*

In the next example, two different sessions with two different users are represented.

% UC
Welcome to UC (Unix Consultant) version 3.23
To a UC '#' prompt, please type in your questions
about the Unix file system in English.
To leave, just type a '^D' or '(exit)'.

Hi.
How can I help you?

How can I delete a file?
Use rm.
For example, to delete the file named foo, type 'rm foo'.

What does rwho do?
Rwho is used to list all users on the network, the users' tty, the users'
login time, and the users' idle time.

% UC
Welcome to UC (Unix Consultant) version 3.23
To a UC '#' prompt, please type in your questions
about the Unix file system in English.
To leave, just type a '^D' or '(exit)'.

Hi.
How can I help you?

How can I find out the inode of a file?
Use ls -i.

What does rwho do?
Rwho is like who, except rwho lists all users on the network.

Because he asked such a simple question, the first user was diagnosed by **KNOME** *as being a novice user. This resulted in the answer to his first question being illustrated by an example, and the extended format of his answer to second. In contrast, the second user asked about a relatively technical aspect of UNIX. Thus, this user is likely to be a relatively sophisticated user. As a result, the answer to that user's first question was not illustrated with an example. For the same reason,* **UC** *presumes that the second user knows about the 'who' command, and takes advantage of this knowledge in generating the response.*

Note this example also illustrates **UC***'s ability to generate different types of responses. This example was able to use a simile format, because the command to be generated was thought to be sufficiently close to one* **UC** *believes the user to already know.*

The next example illustrates some aspects of **UC***'s ability to generate plans.*

How do I print Jim's file on the lineprinter?
Use lpr. This plan will not work if you don't have read
permission on Jim's file.

In this example, **KIP***,* **UC***'s domain planner, suspects that the plan it is suggesting might fail. This is because the plan involves printing someone else's file, and* **KIP** *knows that the lack of appropriate permission in such cases is a likely source of plan failure. However, since it does not know a more certain plan, it delivers the suspect one, but warns the user of the possible problem.*

A fact about a possible problem with a plan is called a concern. Concerns are explicitly represented in **UC***'s knowledge base. At this stage,* **UC***'s knowledge of concerns is fragmentary. Thus, questions above that have the same structure as this one may not cause any warnings to be generating simply because the requisite knowledge is lacking.*

Who is on the system?
I'm sorry, I don't know. Use users to find out who is
on the system.

Here **UC** *fails to answer the user's question, because it does not have any way of running UNIX commands itself. Its failure causes it to apologize.* **UC** *then forms the goal of the user knowing how to determine the requested information. It addresses this goal, which leads to the second part of the response.*

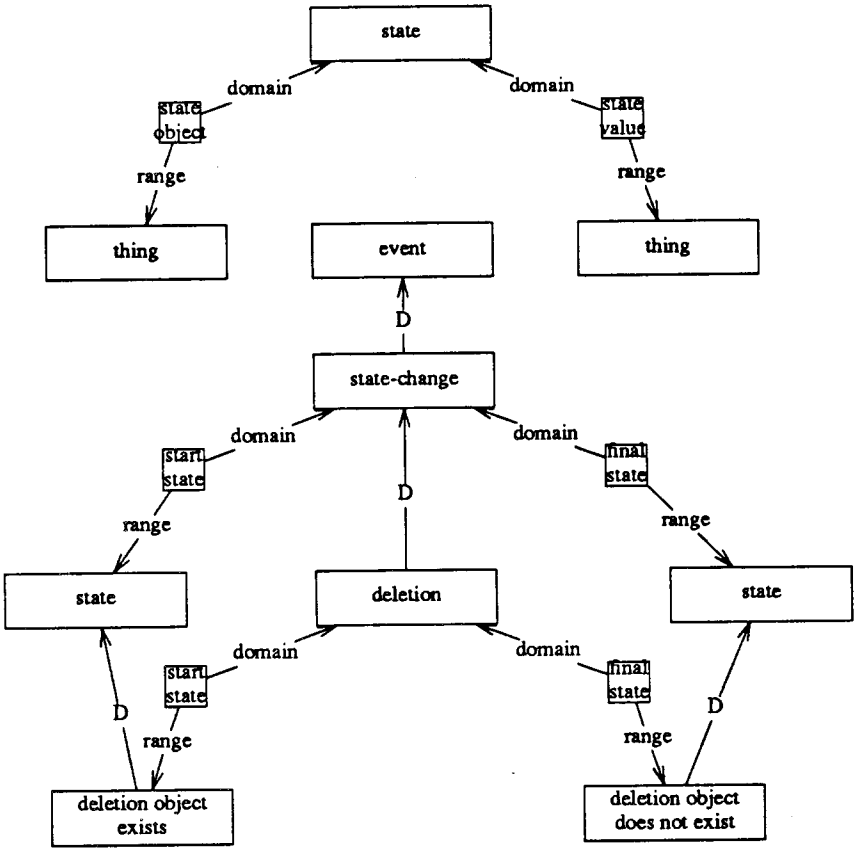

Figure 1.2. KODIAK Examples

This diagram shows the KODIAK definitions of the concepts STATE, STATE-CHANGE, and DELETION. A STATE has a relation for the state object, and one for the value of that object. A STATE-CHANGE is defined in terms of a STATE. DELETION is a subcategory of STATE-CHANGE. This diagram does not depict the manner in which the two states of a STATE-CHANGE are forced to be states of the same object; a mechanism for doing this is described in Chapter 5.

1.2.2. KODIAK

The second main influence on this work is the KODIAK knowledge representation language. Choice of internal representation is the most important influence on a natural language processing system. Because a hierarchical representation with an inheritance mechanism is an absolute necessity if conciseness of representation is to be achieved, I use the KODIAK knowledge representation language. KODIAK is a semantic network representation language that represents facts about the world as a set of concepts, called *absolutes*, related to one another by *relations*. An example of the use of KODIAK in representing knowledge about states, state changes, and deletion is shown in Figure 1.2. Throughout this dissertation, a set of conventions are used for graphically depicting KODIAK objects. Absolutes, such as STATE and DELETION, are drawn as large rectangular boxes. Two absolutes with the same name refer to the same absolute. Relations such

as **START-STATE** and **FINAL-STATE** are drawn as small square boxes. Two relations with the same name represent unique children of the relation with that name. The objects connected by a relation are attached pictorially by arrows emanating from the relation box. These arrows may be labeled with the name of the *aspectual* they represent. An aspectual is in many ways like a slot in a frame system, except that it is a full-fledged **KODIAK** entity. Finally, inheritance in **KODIAK** flows through *dominate* relations, which are drawn as arrows with the label **D**. For a complete discussion of **KODIAK** and the issues that led to its development, see Wilensky [1987a].

1.3. Contributions of This Work

There are four main contributions that are made by this thesis:

1. A theory of what it means to be a good explanation
2. A representation that solves problems with traditional representations
3. A detailed analysis of ambiguity and how it can be handled
4. A method for determining how much processing should be done

First, I present a theory of how an explanation of an utterance may be judged as to its merits as an explanation. I propose three criteria for making such judgements:

1. Applicability
2. Grounding
3. Completeness

The first criterion is the *applicability* of the explanation to the needs of the system that will use it. The applicability criterion is further divided into three types of applicability: applicability of *composition*, which states that the *types* of components that form an explanation must be applicable; applicability of *content*, which holds that each of the components of a particular explanation must be applicable; and applicability of *granularity*, which states that an explanation must contain the appropriate level of detail. The second criterion is the *grounding* of the explanation in what is already known of the speaker and of the dialogue. Finally, the third criterion is the *completeness* of the explanation's coverage of the goals that motivated the production of the utterance. There are two aspects of the completeness criterion: *depth* completeness is the completeness of a single chain of inferences about an utterance; and *breadth* completeness is coverage of all aspects of the utterance. An explanation of an utterance is a good explanation of that utterance to the extent that it meets these three criteria. In addition to forming the basis of a method for evaluating the merit of an explanation, these criteria are useful in designing a goal analysis algorithm; this is because they specify the form that the output of the algorithm must take. Chapter 3 provides a detailed look at these criteria.

The second main contribution of this work is the introduction of a representation for knowledge about plans and goals that solves two major problems inherent in many of the previous approaches to plan and goal representation. The first of these problems, the *intended-effect problem*, arises because there is usually an inadequate distinction made

between the effects of an action and *the* effect that the action was intended to produce. The second problem, the *precondition problem*, arises because the traditional notion of a precondition fails to distinguish between two different types of knowledge: heuristic planning knowledge and knowledge of the defining properties of actions. The solution to these problems, called a *planfor*, is a relation between a type of goal and a sequence of hypothetical actions (called a plan) that constitutes a possible method of achieving a goal of that type. This representation is the subject of Chapter 4.

The third contribution of this work is a detailed analysis of ambiguity. This analysis is divided into three parts. First, Chapter 5 describes how a single explanation of an action can be found. Chapter 6 then discusses the sources of ambiguity. These sources include widely-studied ambiguities at the sentence and utterance levels (lexical ambiguity, structural ambiguity, and referential ambiguity), ambiguity arising from the existence of multiple explanatory plan schemas at the plan recognition level, and two additional categories of ambiguity that occur at the utterance level. The first of these, called *categorization ambiguity*, arises when it is possible to place a given concept into more than one competing category in the knowledge hierarchy. The second, called *topic ambiguity*, arises when it is unclear as to which concept is the topic of a particular relation (such as the topic of a question). Chapter 7 deals with ways to handle these types of ambiguity. Several plan understanding systems have proposed heuristics for disambiguating an utterance once two or more potential explanations for the utterance have been detected. Chapter 7 incorporates these and other heuristics into a generalized framework for handling ambiguity.

The fourth contribution of this work is to introduce a method for determining how much processing of an utterance should be done. There have been two traditional approaches to this problem. Some systems discontinue processing when and only when a particular condition holds. This condition may be related to the explanation itself (as in 'continue processing until a domain goal has been inferred'), or it may be related to the understanding process (as in 'discontinue processing when an ambiguity is encountered'). The other traditional approach is to continue processing until no more inferences can be made. In Chapter 8, I present an algorithm for determining when to continue processing that allows greater flexibility in making its decisions than could be accommodated by previous approaches.

1.4. An Overview of the Goal Analysis Algorithm

The traditional approach to plan recognition, found in seminal works by Wilensky[1978] and Allen [1979] is to chain together a sequence of abductive inferences. PAGAN operates within this paradigm, using planfors to suggest such inferences. The basic goal analysis algorithm has three steps:

1. Find potential explanations of the speaker's utterance
2. Resolve any ambiguities
3. Determine whether to continue the analysis

First, potential explanations of the speaker's utterance must be found. Explanations may be found either among the actions that **PAGAN** is expecting to observe (in which case, the algorithm terminates), or among the explanation schemas (planfors) that **PAGAN** has stored in long term memory. The second step, when more than one interpretation of the utterance is extant, is to try to resolve the ambiguity. Thirdly, **PAGAN** must determine whether it should look for a continuation of the explanation by explaining the goal just inferred. If it decides to do so, these three steps are applied recursively to the inferred goal. Because the recursive step(s) do not differ significantly from the steps taken to analyze the utterance itself, any reference to explanation of an utterance within the remainder of this thesis should be taken to apply both to the explanation of utterances, and to the explanation of inferred goals. A diagram of the flow of control of the algorithm is shown in Figure 1.3. The steps of the algorithm, and the way in which those steps build explanations that meet the criteria for good explanations, are explained in more detail in the following sections.

1.4.1. Step Zero: Interpreting the Utterance

Before goal analysis can begin, the user's utterance must be read and converted to an internal representation. This process produces one or more interpretations of the user's utterance. These interpretations must be marked as being mutually incompatible before being handed to **PAGAN** for goal analysis. This allows **PAGAN** to address ambiguities that prior components of the system have been unable to resolve.

1.4.2. Step One: Finding an Explanation

The first step of the goal analysis algorithm is to find one or more potential explanations of the utterance. This step is described in detail in Chapter 5. There are three places that **PAGAN** may find an explanation for an utterance:

1. Among its expectations of speaker actions
2. Among the abstract planfors in its knowledge base
3. Among the non-intentional explanations in its knowledge base

First, an explanation may be found among the expectations that are held about future actions of the speaker. Such expectation matching builds explanations that meet the grounding criterion, because they relate the utterance to the previous dialogue structure. For example, if the system were expecting an answer to a yes/no question, the statement:

User: No.

could easily be matched against that expectation.

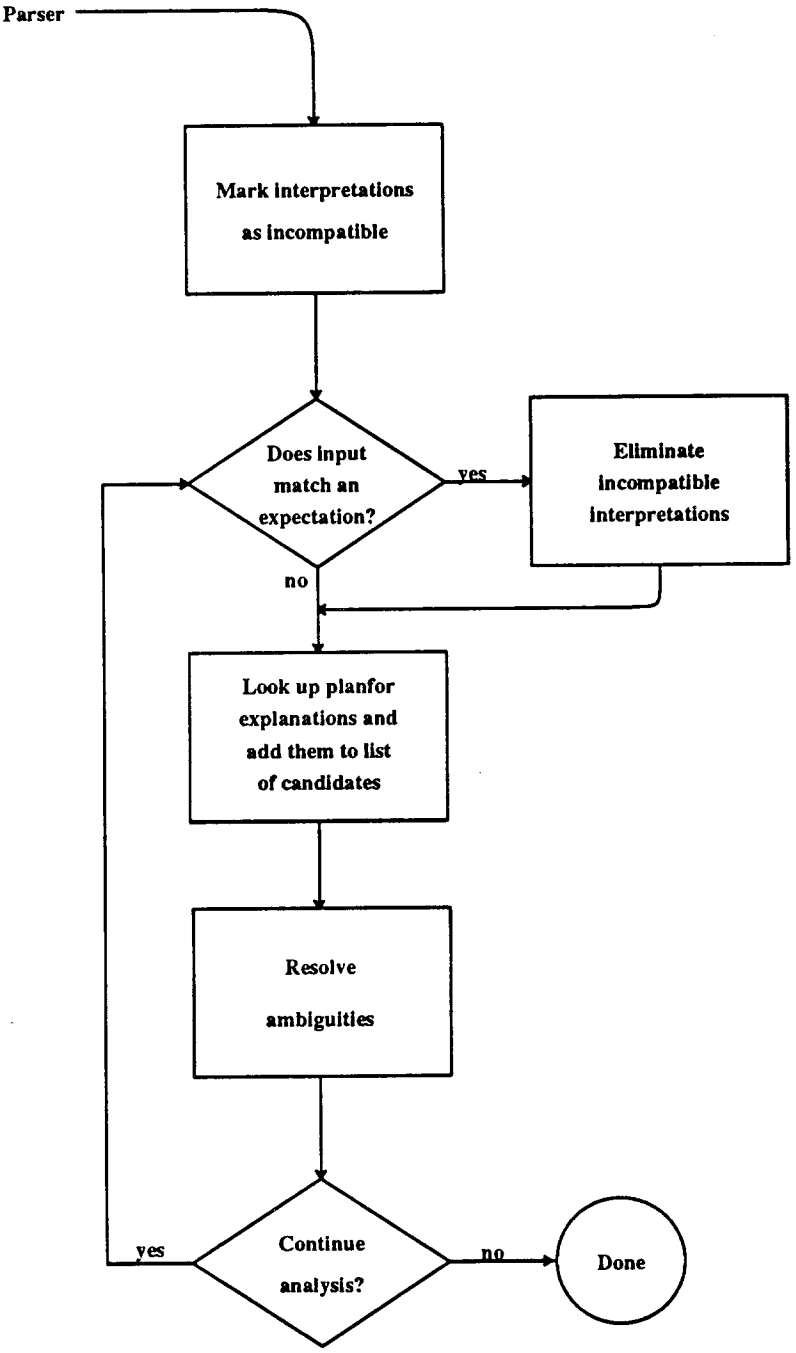

Figure 1.3. The Goal Analysis Algorithm

Secondly, explanations may be found among the collection of abstract plan schemas, which I call *planfors*, stored in the long-term knowledge base. For example, the general notion of a question and answer sequence as a plan for the goal of learning something could provide an explanation for a query such as:

User: How do I copy a file?

Building explanations based on planfors causes them to meet the applicability of composition and the applicability of granularity criteria, to the degree that the planfors in the knowledge base reflect the needs of the system. The use of planfors also aids in building explanations that meet the breadth completeness criterion, since a single utterance might be explained by multiple *compatible* planfors.

Finally, explanations may be found among the set of non-intentional explanations in the knowledge base. For example, the non-intentional explanation 'computer users often want to edit files' might be used to explain why a particular user wants to edit a particular file. The use of a non-intentional explanation does not necessarily assist in building explanations that meet the criteria discussed above, but it does offer a method for terminating a sequence of inferences about an utterance.

1.4.3. Step Two: Handling Ambiguity

The second main step of the goal analysis algorithm is to handle any ambiguities that have arisen. Ambiguity can arise because the sentence used in the utterance is ambiguous, because the use of the sentence is ambiguous, or because the utterance as a whole (or one the goals inferred to explain the utterance) might be part of more than one plan. The various sources of ambiguity are discussed in detail in Chapter 6.

Chapter 7 describes how ambiguities are handled. There are two ways to handle ambiguity:

1. Resolve the ambiguity, and continue processing
2. Ignore the ambiguity, processing each alternative individually

First, the ambiguity can be resolved. Ambiguity resolution can itself be done in two ways:

1. By rejecting all incorrect interpretations, leaving the correct one
2. By selecting the correct interpretation outright

Rejection of an interpretation is done by finding a conflict between some portion of that interpretation and some other knowledge held by the user model. Such conflicts are called *knowledge conflicts*. If the strength of belief in the conflicting knowledge from the user model is greater than the strength of belief in the interpretation of the utterance, then that interpretation can be rejected. For example, if it is known that the user believes that the system has extensive knowledge of UNIX, then the direct interpretation of the question:

User: Can you tell me how to copy a file?

can be eliminated.

The other way that an ambiguity can be resolved is by selecting one of the competing interpretations as the correct one. Such a selection can be made if one of the interpretations matches an expected action, as described above. When such a match occurs, that interpretation is selected as the correct one. All other interpretations that *conflict* with the selected one are then rejected. For example, if it is known that the user wants to free up disk space, then an interpretation of an utterance that presupposes that the user wants to delete a file can be selected at the expense of any rival explanations.

If an ambiguity cannot be resolved, it is ignored. That does not mean that it can never be resolved. Rather, as each interpretation is extended by subsequent iterations of the algorithm, the ambiguity can be re-examined. If **PAGAN** is never able to resolve the ambiguity, it is treated as an ambiguity that should be detected by the system as a whole. Subsequent components of the system are then free to pursue other techniques for handling the ambiguity, such as initiating a clarification subdialogue, or addressing each of the interpretations individually.

1.4.4. Step Three: Extending an Explanation

The third step of the goal analysis algorithm is to determine whether the inferred explanation for the utterance should itself be explained. This decision is based on an assessment of the depth completeness of the explanation chain, as described above. An explanation that is complete is not extended any further. An explanation is complete when it meets some expectation instantiated by previous dialogue, or when the most recently inferred goal lies beyond the system's domain of expertise. An explanation that is incomplete is subjected to further analysis by the algorithm. An explanation is incomplete when the inferred goal is always used in service of some other goal, or when the dialogue model contains a strong expectation that has not yet been met. Thus, the third step of the algorithm helps to build explanations that meet the applicability of content criterion.

If none of these conditions holds, then it is not immediately clear whether the explanation is complete. In such cases, two additional assessments are made to determine whether processing should continue. First, the system determines its level of curiosity about the explanation. This is compared against an estimate of the difficulty of continuing the analysis. If the level of difficulty is higher, processing is terminated. In such cases, the production of a good explanation is sacrificed in order to conserve system resources. On the other hand, if the level of curiosity about the explanation is higher, processing continues.

1.5. An Example

This section shows the main aspects of PAGAN's processing of an utterance, via a commented trace. First, I describe the use of traces throughout this dissertation. Then I step through such a trace. The trace provides a taste of the theory espoused in this dissertation, and illustrates some of the issues that arise in the implementation of a goal analyzer. Finally, a second trace shows how PAGAN's processing fits into the processing performed by UC as a whole.

1.5.1. Reading The Traces

Throughout this dissertation, the theories I put forth are illustrated with trace output of the PAGAN program. The following font conventions are used in the traces. KODIAK objects are written in BOLD CAPITALS. The sentence that PAGAN is processing is shown at the beginning of each trace, and is written in (BOLD CAPITALS) surrounded by parentheses within the body of the trace. Descriptions produced by PAGAN as it runs are written in roman, and comments added by hand after-the-fact are written in *italics*. In some places, it is illuminating to depict a portion of the representation network produced by PAGAN as it processes an utterance. Since it is difficult to convert such networks to a pictorial representation automatically, I use a method that converts them to LISP lists; such lists are then printed in linear form. Each list is composed of the name of the object depicted in the list, followed by a description of the relations in which the object takes part. This method is imperfect, and does not always show exactly and only the portion of the network that would best illustrate the point being made. Nevertheless, it does give a fair idea of the structures being built; I try to rectify any deficiencies in this output with the addition of appropriate comments.

In general, traces only show that portion of PAGAN's processing relevant to the topic being discussed. In particular, no ambiguity is introduced in the traces in the chapters that do not themselves address ambiguity. This is done by restricting PAGAN's knowledge base in those examples to the facts that are necessary for the desired inferences to be made. In the chapters that address ambiguity, the full range of knowledge is made available to PAGAN in the traces. Some traces also terminate before PAGAN's processing is complete, because the point they are designed to illustrate has been made. This is done by removing the extraneous trace information after-the-fact. Other than deletions such as these, and the introduction of comments in italics, the only changes made to PAGAN's trace output by hand are changes in appearance, such as introduction of font information and the addition of white space.

Two naming conventions should be noted. First, names that end with a number represent subconcepts of the concept represented by the root name. For example, LOCATION-OF8 lies below the LOCATION-OF concept in the knowledge hierarchy. Secondly, each KODIAK relation has an inverse; a name that ends with an asterisk represents the inverse of the relation represented by the root name. For example, if LOCATION-OF8 is a relation that holds from FILE15 to the /tmp-directory concept, then

its inverse, called LOCATION-OF8*, holds from the /tmp-directory concept to FILE15.

1.5.2. A Sample Trace

This trace shows PAGAN's processing of one of the utterances from the faulty dialogue at the beginning of this chapter:

User: What is a way to delete a file?

As demonstrated in that dialogue, this utterance is ambiguous. The trace shows the detection of the ambiguity, and its subsequent resolution. Following the trace of PAGAN's processing of this question is a trace of UC's processing of the question. It includes not only PAGAN's processing, but also the processing done by the KIP planner and by the generation mechanism.

User: What is a way to delete a file?

The parser produced the following interpretations:

The semantic interpreter produces two interpretations of this question. The two are identical, with the exception of the category into which the question itself is placed. In the first interpretation, the utterance is listed as a SPECIFICATION-QUESTION. *This means that the user is expecting to be told how to delete a file. This is the correct interpretation, and the one that will eventually be selected by* PAGAN.

Interpretation 1 (TELL-ACTION73):
(HAS-GOAL159 (DOMAIN USER)
 (RANGE
  (TELL-ACTION73 (DOMINATED BY ASK-SPECIFICATION-QUESTION)
   (ACTOR85 = USER) (HEARER62 = UNIX-CONSULTANT) (SPEAKER57 = USER)
   (UTTERANCE64 =
    (SPECIFICATION-QUESTION16 (DOMINATED BY SPECIFICATION-QUESTION)
     (WHAT-IS51 =
      (DELETE-ACTION28 (DOMINATED BY DELETE-ACTION)
       (CAUSES-OF-DELETE-ACTION35 =
        (DELETION42 (DOMINATED BY FILE-DELETION)
         (STATE-CHANGE-OBJECT36 = (FILE41 (DOMINATED BY FILE)))))))))))

In the second interpretation, the utterance is listed as a DESCRIPTION-QUESTION. *This means that the user is expecting to be told what it means to be a way to delete a file. This is the interpretation adopted by the faulty goal analyzer in the dialogue at the beginning of this chapter; it will be rejected by* PAGAN *later in the trace.*

Interpretation 2 (TELL-ACTION74):
(HAS-GOAL160 (DOMAIN USER)

(RANGE
(TELL-ACTION74 (DOMINATED BY ASK-DESCRIPTION-QUESTION)
(ACTOR86 = USER) (HEARER63 = UNIX-CONSULTANT) (SPEAKER58 = USER)
(UTTERANCE65 =
(DESCRIPTION-QUESTION9 (DOMINATED BY DESCRIPTION-QUESTION)
(WHAT-IS52 =
(DELETE-ACTION29 (DOMINATED BY DELETE-ACTION)
(CAUSES-OF-DELETE-ACTION36 =
(DELETION43 (DOMINATED BY FILE-DELETION)
(STATE-CHANGE-OBJECT37 = (FILE42 (DOMINATED BY FILE)))))))))))

Since the semantic interpretation of the user's utterance is treated as an explanation of that utterance, PAGAN*'s processing begins with step two of the goal analysis algorithm: handling ambiguity. Unfortunately, although the interpretation being considered here is the incorrect interpretation, the ambiguity cannot be resolved at this point.*

Attempting to resolve any ambiguities in TELL-ACTION74
Retrieving potential conflicts with TELL-ACTION74 and its plan.
No conflict found.

PAGAN *now determines whether it should continue its analysis of the incorrect interpretation. As you can see,* PAGAN *performs steps two and three of the algorithm on a single interpretation before moving on to the next interpretation. The goal that* PAGAN *is considering at this point (*TELL-ACTION74*) is found to be an instrumental goal. This means that it is something that would only be done in service of some other goal. Therefore,* PAGAN *decides to continue its processing of this interpretation.*

Determining whether to chain on TELL-ACTION74.
Determining whether TELL-ACTION74 is a complete explanation of
(WHAT IS A WAY TO DELETE A FILE).
Determining whether TELL-ACTION74 is outside of UC's domain of expertise.
TELL-ACTION74 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether TELL-ACTION74 is an incomplete explanation of
(WHAT IS A WAY TO DELETE A FILE).
Determining whether TELL-ACTION74 is an instrumental goal.
TELL-ACTION74 is an instrumental goal.
The explanation is incomplete.
Chaining should be done on TELL-ACTION74.

Now PAGAN *goes back and performs steps two and three of the algorithm on the other interpretation of the user's utterance. The analysis here is virtually identical to the previous analysis.*

Attempting to resolve any ambiguities in TELL-ACTION73
Retrieving potential conflicts with TELL-ACTION73 and its plan.
No conflict found.

Determining whether to chain on TELL-ACTION73.
Determining whether TELL-ACTION73 is a complete explanation of
(WHAT IS A WAY TO DELETE A FILE).
Determining whether TELL-ACTION73 is outside of UC's domain of expertise.
TELL-ACTION73 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether TELL-ACTION73 is an incomplete explanation of
(WHAT IS A WAY TO DELETE A FILE).
Determining whether TELL-ACTION73 is an instrumental goal.
TELL-ACTION73 is an instrumental goal.
The explanation is incomplete.
Chaining should be done on TELL-ACTION73.

PAGAN *now recursively analyzes the two interpretations, starting from step one of the algorithm. First, it looks for an explanation of the correct interpretation. Since there is no previous dialogue in this example, there is no opportunity to ground the utterance. So,* PAGAN *tries to find a planfor explanation. It finds one; this planfor states that to know something, ask about it and get an answer to the question.* PAGAN *infers that the user may have the goal of knowing how to delete a file. This goal is called* KNOWING-THAT30.

Attempting to find an explanation for TELL-ACTION73.
Determining whether TELL-ACTION73 is grounded in the preceding dialogue.
TELL-ACTION73 was not grounded in the preceding dialogue.

Trying to find planfor explanations for TELL-ACTION73.
Found PLANFOR4 as an explanation for TELL-ACTION73.
Synopsis:
Goal: know something;
Step 1: ask;
Step 2: be told.
Inferred goal is KNOWING-THAT30.

Explanation found for TELL-ACTION73.

PAGAN *must also find an explanation of the incorrect interpretation. Again, there is no previous dialogue to support grounding.* PAGAN *does find an appropriate planfor though; this planfor states that to know the meaning of something, ask what its meaning is, then receive a reply.* PAGAN *infers that the user may have the goal of knowing the meaning of 'a way to delete a file.' This goal is called* KNOWING-MEANING5.

Attempting to find an explanation for TELL-ACTION74.
Determining whether TELL-ACTION74 is grounded in the preceding dialogue.
TELL-ACTION74 was not grounded in the preceding dialogue.

Trying to find planfor explanations for TELL-ACTION74.
Found PLANFOR5 as an explanation for TELL-ACTION74.
Synopsis:
Goal: know the meaning of something;
Step 1: ask;
Step 2: be told.
Inferred goal is KNOWING-MEANING5.

Explanation found for TELL-ACTION74.

> PAGAN *now proceeds to step two of the algorithm. It tries to determine whether there is any conflict between the inferred goal* KNOWING-MEANING5 *and knowledge of the user maintained by the user model. In this case, there is such a conflict. For this interpretation to be valid, one would have to assume that the user did not already know what it meant to be a way to delete a file. However, the user model indicates that this is an unlikely assumption. Therefore, this interpretation of the user's utterance is rejected.*

Attempting to resolve any ambiguities in KNOWING-MEANING5
Retrieving potential conflicts with KNOWING-MEANING5 and its plan.
Conflict found: inferred goal already obtains.
Rejecting KNOWING-MEANING5 as an explanation of TELL-ACTION74.

> *Only the correct interpretation remains.* PAGAN *now determines whether there is any reason to disbelieve this interpretation. No such reason is evident, so* PAGAN *goes on to decide whether processing should continue on the interpretation.* KNOWING-THAT30 *is found to be an instrumental goal, so processing continues.*

Attempting to resolve any ambiguities in KNOWING-THAT30
Retrieving potential conflicts with KNOWING-THAT30 and its plan.
No conflict found.

Determining whether to chain on KNOWING-THAT30.
Determining whether KNOWING-THAT30 is a complete explanation of TELL-ACTION73.
Determining whether KNOWING-THAT30 is outside of UC's domain of expertise.
KNOWING-THAT30 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether KNOWING-THAT30 is an incomplete explanation of TELL-ACTION73.
Determining whether KNOWING-THAT30 is an instrumental goal.
KNOWING-THAT30 is an instrumental goal.
The explanation is incomplete.
Chaining should be done on KNOWING-THAT30.

> PAGAN *now tries to find the reason that the user wants to know how to delete a file. It finds an appropriate planfor, which states that knowing how to do something is a step toward actually doing it. Thus,* PAGAN *infers that the user may have the goal of actually deleting a file. This goal is represented by* DELETION45.

Attempting to find an explanation for KNOWING-THAT30.
Determining whether KNOWING-THAT30 is grounded in the preceding dialogue.
KNOWING-THAT30 was not grounded in the preceding dialogue.

Trying to find planfor explanations for KNOWING-THAT30.
Found PLANFOR8 as an explanation for KNOWING-THAT30.
Synopsis:
Goal: Achieve X;
Step 1: Know how to X;
Step 2: Do X
Inferred goal is DELETION45.

Explanation found for KNOWING-THAT30.

> *Step two of the algorithm finds no reason to believe that* DELETION45 *is a poor interpretation of the user's utterance.*

Attempting to resolve any ambiguities in DELETION45
Retrieving potential conflicts with DELETION45 and its plan.
No conflict found.

> *Step three of the algorithm is now invoked on* DELETION45. *This time, the estimated cost of continuing the analysis exceeds the expected usefulness of continuing the analysis. Therefore, the algorithm halts at this point.* PAGAN *makes one last attempt to find a general thematic explanation of why the user would want to delete a file, but finds none. This terminates* PAGAN*'s processing of the utterance.*

Determining whether to chain on DELETION45.
Determining whether DELETION45 is a complete explanation of KNOWING-THAT30.
Determining whether DELETION45 is outside of UC's domain of expertise.
DELETION45 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether DELETION45 is an incomplete explanation of KNOWING-THAT30.
Determining whether DELETION45 is an instrumental goal.
DELETION45 is not an instrumental goal.
Determining whether there is a strong expectation for a particular action.
There is no strong expectation for a particular action.
The explanation is not necessarily incomplete.
Determining the level of curiosity about DELETION45.
Determining the level of interest in DELETION45.
Interest level = 0.
Determining the level to which the goal DELETION45 conflicts with system goals.

Conflict level = 0.
Curiosity level = 0.
Determining estimated cost of chaining on DELETION45.
Estimated cost = 1.
Chaining should not be done on DELETION45.

Determining whether DELETION45 is explained by a theme.
DELETION45 was not explained by a theme.

The following trace places PAGAN's processing in context, by showing UC's successful processing of the question from the previous trace:

User: What is a way to delete a file?

This trace is based on an earlier version of PAGAN than is described in the rest of this dissertation. This version is written in Franz Lisp rather than Common Lisp. It places a premium on speed, so as to allow UC to respond to an average question in 6-10 seconds. Consequently, it does not support the entire range of abilities suggested herein. However, the overall philosophy, as well as many of the implementation details, are shared by the two versions.

Welcome to UC (Unix Consultant) version 3.23
To a '#' prompt, please type in your questions
about the UNIX file system in English.
To leave, just type '^D' or '(exit)'.

Hi.
How can I help you?

What is a way to delete a file?

The parser produces the single interpretation of the question that is most likely to reflect the user's intentions. This speeds goal analysis at the expense of occasionally misinterpreting the user's question. The representation of the user's question is shown as a set of descriptions; however, internally, these descriptions are interrelated.

The parser produces:

```
(ASK10
 (listener10 = UC)
 (speaker10 = *USER*)
 (asked-for10 =
  (QUESTION10 (what-is10 = (KNOW61?
                    (fact61 = (ACTION14?
                                (actor14 = PERSON39?)))
```

```
                    (knower61 = UC))))))
(DELETE-ACTION0?
 (del-effect0 = (DELETE-EFFECT0?
                 (DELETE-EFFECT-final-state0 =
                  (EXISTS0 (exist-object0 = FILE3?)
                           (existence0 = FALSE)))
                 (DELETE-EFFECT-initial-state0 =
                  (EXISTS3 (exist-object3 = FILE3?)
                           (existence3 = TRUE)))
                 (del-object0 = FILE3?)))
 (cause0-0 = (ACTION14? &)))
```

PAGAN *is called, and handed as its argument the representation of the user's question.* **PAGAN** *determines that the user probably wants to know how to delete a file (as represented by 'ACTION14?').*

The goal analyzer produces:

```
((HAS-GOAL-ga0
  (planner-ga0 = *USER*)
  (goal-ga0 = (KNOW-ga0?
               (knower-ga0 = *USER*)
               (fact-ga0 = (ACTION14? &))))))
```

The **KIP** *planner is now given the task of figuring out how to delete a file.*

The planner is passed:

```
((DELETE-EFFECT0? &))
```

The planner produces the plan of using the UNIX **rm** *command.*

The planner produces:

```
(PLANFOR250
 (goals250 = (DELETE-EFFECT0? &))
 (plan250 = (UNIX-RM-COMMAND0
             (rm-file0 = FILE3?)
             (UNIX-RM-COMMAND-effect0 = (DELETE-EFFECT0? &)))))
(HAS-FILE-NAME19 (named-file19 = FILE3?)
                 (file-name19 = (lisp= nil)))
(RM-HAS-FORMAT0
 (RM-HAS-FORMAT-command0 = (UNIX-RM-COMMAND0 &))
 (RM-HAS-FORMAT-format0 = (RM-FORMAT0
                           (rm-file-arg0 = file-name19)
                           (RM-FORMAT-step0 = rm))))
(HAS-COMMAND-NAME00
 (HAS-COMMAND-NAME-named-obj00 = (UNIX-RM-COMMAND0 &))
 (HAS-COMMAND-NAME-name00 = rm))
```

UC *sends* **KIP**'*s plan, together with an example of the use of that plan, to the generator.*

The generator is passed:

```
(TELL5 (effect5 = (STATE-CHANGE1 (final-state1 = (KNOW-ga0? &))))
    (listener5-0 = *USER*)
    (speaker5-0 = UC)
    (proposition5 = (PLANFOR250 &)))
```

The generator is passed:

```
(TELL6
 (speaker6-0 = UC)
 (listener6-0 = *USER*)
 (proposition6 =
  (EXAMPLE0
   (example0 =
    (PLANFOR250-0
       (goals250-0 =
        (DELETE-EFFECT0-0?
         (DELETE-EFFECT-final-state0-0 =
          (EXISTS0-0
           (exist-object0-0 = FILE3-0?)
           (existence0-0 = FALSE-0)))
         (del-object0-0 = FILE3-0?)
         (DELETE-EFFECT-initial-state0-0 =
          (EXISTS3-0
           (exist-object3-0 = FILE3-0?)
           (existence3-0 = TRUE-0)))))
       (plan250-0 =
        (TYPE-ACTION0
         (speaker0-4 = *USER*)
         (type-string0 =
          (RM-FORMAT0-0
           (rm-file-arg0-0 =
            (file-name19-0 = aspectual-of
                        (HAS-FILE-NAME19-0
                           (named-file19-0 = FILE3-0?)
                           (file-name19-0 = foo))))
           (RM-FORMAT-step0-0 = rm)))))))))
```

Finally, UC*'s response is produced.*

Use rm.
For example, to delete the file named foo, type 'rm foo'.

Chapter 2

Previous Research

In this chapter, I give an overview of previous research in goal analysis and related topics. My aim in this chapter is to present the fundamental work in two topics central to my research:

1. Plan recognition
2. Dialogue models and user models

Plan recognition is the task of inferring plans and goals from observed actions. Dialogue models are models of the status of a dialogue between two or more participants. Dialogue models may contain a record of the utterances used in the dialogue, and of the objects that are currently at the forefront of the dialogue. User models are models of the dialogue participants themselves. They may include assessments of overall knowledge levels, as well as individual facts and beliefs held by the participants.

2.1. Plan Recognition

Plan recognition is the task of inferring, based on observed actions, the plans that led the agent or agents involved to perform those actions. Goal analysis is simply plan recognition in which the emphasis is placed on inferring the goals of the agent. Plan recognition can be divided into two categories:

1. Intended recognition
2. Unintended recognition

Intended recognition is the kind of recognition required in dialogue processing. In intended recognition, the speaker makes an utterance and *intends* that the hearer infer the motivating plans and goals.

The other kind of plan recognition, which brings with it a different set of concerns, is unintended recognition. In unintended recognition, an observer attempts to infer the plans and goals of an agent who is unaware of or unconcerned with the observer. For example, one of the components of the SINIX Consultant [Kemke 1986, Hecking et al. 1988] is a plan recognizer called **REPLIX** [Hecking 1987]. **REPLIX** monitors the communication between a user and the SINIX operating system. The purpose of **REPLIX** is to interpret a sequence of commands to SINIX as one or more plans aimed at achieving particular goals. **REPLIX** can then suggest an improved plan when an inefficiency in the user's plan is detected. **REPLIX** also has the ability to recognize embedded and interleaved plans. As long as the user's plans are deemed to be efficient, **REPLIX** behaves as if it did not exist. Thus, **REPLIX** performs unintended plan recognition.

Unintended recognition is more difficult than intended recognition, because an agent who intends to have his or her plans and goals recognized will provide clues and obey conventions that facilitate communication. My work has been concerned primarily with intended recognition. However, the boundary between intended recognition and unintended recognition is a fuzzy one, and **PAGAN** will often attempt to make inferences that the speaker did not necessarily intend.

One branch of plan recognition that has proven fertile is story understanding. **SAM** (Script Applier Mechanism) used the notion of a stereotyped sequence of events, called a script, to perform story understanding [Schank and Abelson 1977]. A script is simply a template for a commonly-occurring sequence of events. For example, a script for eating at a restaurant might include specifications for entering the restaurant, ordering, eating, and leaving the restaurant. Story understanding is performed by matching described actions against the action descriptions in the script. Scripts are useful because they allow rapid interpretation of stereotypical sequences of events, and allow events that are part of that sequence but that are not observed to be inferred.

One of the major problems with the **SAM** approach was that it was only useful in understanding the most mundane sorts of stories. Any story that did not conform to a highly stereotypical sequence of events could not be handled by **SAM**, because **SAM**'s scripts were designed specifically to handle common plans. To address these difficulties, Wilensky developed the **PAM** system [1978, 1983]. **PAM** examined simple stories to determine the plans and goals of the characters within those stories. Instead of viewing a scene of the story as a single unit, as in the **SAM** approach, **PAM** broke down its explanations of the story into units at the action level. An example of a story fragment handled by **PAM** is the following:

> *Willa was hungry.*
> *She picked up the Michelin Guide and got into her car.*

PAM could process this story, then use its analysis to answer questions about (among other things) Willa's motivation for her actions.

PAM used an approach to plan and goal recognition called *explanation-driven understanding*. The idea was to find an explanation, in terms of plans, goals, and themes, for every observed or inferred action, plan, or goal. In Wilensky's system, an action was explained by the plan it instantiated, a plan was explained by the goal the plan was designed to achieve, and a goal was explained either by the theme that gave rise to it, or by a plan for which its achievement was instrumental. For example, PAM's processing of the last sentence in the above story fragment went along the following lines. The action of picking up the guide is a plan for possessing it, so Willa may have the goal of possessing it. Possessing an object is a plan for the goal of using it, so Willa may have the goal of reading the guide. Reading the guide is a plan for knowing the information it contains, and since the Michelin Guide contains information about restaurants, Willa may have the goal of knowing the location of a restaurant. Eventually, by reasoning along these lines, PAM is able to determine that Willa may want to go to the restaurant so as to eat. Since this interpretation meshes with the goal of satisfying hunger predicted during the analysis of the first sentence of the story, PAM's processing halts. Any competing interpretations, such as reading for pleasure, are discarded.

The treatment of language as action was first pursued by philosophers of language [Austin 1962, Searle 1969]. One of the earliest works to apply Searle's approach to speech acts to the task of computerized language understanding was done by Allen [1979, Allen and Perrault 1980]. Allen's system was designed to simulate a clerk at a train station. The user could ask the system simple questions about train arrivals and departures. Here is an example of a simple exchange whose corresponding representations the system could successfully process:

> **Patron**: When does the train to Windsor leave?
> **System**: The train leaves at 1600.
> **System**: The train leaves from gate 7.

In this example, the patron asks a simple direct question. The system provides the answer to the patron's question, and also volunteers additional information about the departure gate. It selects this additional information by analyzing potential obstacles to the patron's inferred plan, and acting to overcome those obstacles.

Allen's system was designed to view language as goal-directed action. Thus, speech acts were represented as plans. For example, the following plan was used to handle a request, either direct or indirect:

> **REQUEST**(speaker,hearer,action)
> body:
> **MB**(hearer, speaker, speaker **WANT** hearer **DO** action)
> effect:
> hearer **WANT** hearer **DO** action

This plan says that a request is done by making it mutually believed that the speaker wants the hearer to do some action. The effect of the request is that the hearer wants to do the action.

Allen developed a set of inference rules for building explanations. The most important of these rules are:

1. The Precondition-Action Rule
2. The Body-Action Rule
3. The Action-Effect Rule

The Precondition-Action Rule states that if a speaker wants to achieve a particular goal, the speaker may want to achieve some other goal enabled by it. The Body-Action Rule states that if a speaker wants to execute some action, the speaker may want to execute some other action of which the first action forms one part. The Action-Effect Rule states that if a speaker wants to execute some action, the speaker wants the effects of that action to hold. Any plan recognition algorithm must take these three types of inference into account. In Chapter 4, I show how the knowledge needed to make these three types of inference can be combined into a single representation.

There are a number of problems with Allen's formulation of the plan recognition process. Some of these problems are addressed individually in later chapters. However, the basic approach is compelling, and forms the basis for much of the subsequent work in plan understanding.

Allen and many other researchers have been concerned not only with inferring the goals that underlie the production of an utterance, but also with the satisfaction of those goals. I am concerned only with the former. Satisfaction of the user's goals in UC is performed by the **UCEgo** agency mechanism [Chin 1988] and by the **KIP** planner [Luria 1988].

Wilensky [1983] introduced the notion of a *metaplan* for use by a planning agent. A metaplan is a plan that manipulates another plan. Metaplans are useful in goal analysis because speakers' utterances are often aimed at clarifying or otherwise using plans. This is especially true in systems that operate in instructional or information-seeking domains. If utterances are viewed as components of plans, then utterances about plans are effectively components of metaplans.

Litman [1985] used this notion of a metaplan to expand Allen's approach to dialogue processing. Her system integrates plan recognition with discourse analysis. As in Allen's system, Litman's system could interpret an utterance as the continuation of an existing plan. However, Litman's system also allowed an utterance to introduce a metaplan to an existing plan. For example, Litman's system could engage in the following dialogue (taken from Litman and Allen [1984]):

Passenger: The eight-fifty to Montreal?
Clerk: Eight-fifty to Montreal. Gate seven.

In the analysis of the passenger's question, Litman's system creates a stack of plans. The bottom plan in the stack is the **BOARD** plan, which involves boarding the train to Montreal. Above this plan on the stack is a meta-plan for finding out one of the parameters of

the BOARD plan, namely the location of the boarding. The use of metaplans allows this system to handle a wider range of dialogues than could Allen's system, while producing structures that more accurately reflect the relationships among the inferred plans.

Actually, the use of the term 'metaplan' for a plan that relates to another concept in this way is somewhat misleading. This term was adopted because much of the recent research in dialogue processing has been done in planning-oriented domains. Thus, this sort of plan typically has another plan as the object it manipulates, and is therefore properly called a metaplan. However, the system should work regardless of whether the item on the stack to which the utterance is attached is a plan. Consider for example the following dialogue fragment:

> User: What's a file descriptor?
> UC: It's a magic cookie for input/output.
> User: What's a magic cookie?

It is a stretch to call the *content* of UC's utterance being questioned by the user (that is, the term 'magic cookie') a 'plan' or a 'plan parameter,' even though UC's utterance as a whole is certainly part of a plan. Therefore, the term 'metaplan' is inappropriate in this example.

Carberry [1983, 1986a] built a system called TRACK, which interprets utterances based on questions a user may have about a plan she is trying to construct. As with UC, TRACK's primary area of application is information-seeking dialogues -- dialogues in which one agent is trying to obtain information from another agent. Given a stated goal, TRACK attempts to infer other goals held by the speaker through the use of a set of heuristics. For example, if the speaker asks "What are the prerequisites of History 304?," then the speaker may have the goal of taking the history class. Given a particular goal, TRACK goes on to choose possible domain plans that the speaker may have in mind. Again there are a set of heuristics for locating such a plan (i.e. for indexing such a plan in memory). For example, if the goal is a true predicate which is a precondition to or step in a plan, then that plan may be the domain plan with which the user is currently concerned. Using these techniques, TRACK builds a tree structure representing the domain-level plan the speaker is pursuing. It then uses a set of rules of conversation to fit new utterances into this tree structure. TRACK is particularly useful in solving this latter problem. For example, TRACK handles inter-sentential ellipsis particularly nicely. My work contrasts with TRACK in that I adopt a uniform approach to inferring both linguistic goals and domain goals.

An important area of plan recognition that my work does not address is the detection of speaker misconceptions. The treatment of speaker misconceptions is the subject of much current research. Pollack [1984] identifies the *appropriate query assumption* as the assumption that a speaker always understands what advice is needed, and asks for that advice correctly and directly. Pollack then explores plan recognition when the appropriate query assumption is not made. I make the appropriate query assumption; however, the degree to which this assumption restricts the dialogue is lessened in PAGAN by using the user model to mediate selection of the correct interpretation of an utterance,

and by the development of a network of explanatory goals for an utterance instead of just a single such goal.

Quilici et al. [1985] were concerned with another type of failure -- incorrect user domain plans. Quilici identified nine problem classes that might cause a plan to be incorrect, and associated with each a set of heuristics for giving advice about a problem in that class. See Luria [1988] for a description of how UC's planner handles some of these issues.

The plan recognition techniques described above are primarily backward chaining techniques. It is also possible to use forward chaining for plan recognition. For example, the **BELIEVER** system [Schmidt et al., 1978] relied heavily on a predictive component for understanding simple action descriptions. The advantage of using forward chaining for plan understanding is that, when successful, this approach greatly reduces the amount of work required to recognize a plan once a step of that plan has been observed. The disadvantage of forward chaining is that when it is unsuccessful, the system must backtrack before recognition can be completed. **BELIEVER** devoted much of its energies to hypothesis revision, in order to correctly handle situations in which the observed action had not been predicted. Because of this drawback of the forward chaining paradigm, **PAGAN** uses backward chaining almost exclusively.

One type of forward-chaining inference that **PAGAN** does use is based on work by Norvig [1987]. Norvig examined the low-level inferences that a reader easily makes in reading a story. For example, on reading 'In a poor fishing village built on an island not far from the coast of China,' a reader will typically infer that the villagers fish in the sea, and that this same sea surrounds the island, and forms the coast of China. Norvig's approach is to use marker passing to suggest low-level inferences. One such type of inference used by **PAGAN**, called a *concretion inference*, is discussed in Chapter 5.

2.2. Dialogue Models and User Models

Two extremely important components of a plan recognition system are the dialogue model and the user model. There is some disagreement in the literature as to the relative scope of these two models. Kobsa and Wahlster [1987] provide a good summary of the various viewpoints. I take the position in this dissertation that the dialogue model and the user model are separate entities. In my formulation, the dialogue model contains a representation of the structure of the dialogue and of the plans and goals that motivate the dialogue, as well as a description of the objects that are currently in focus. The user model on the other hand contains a description of the user's knowledge and beliefs independent of the dialogue (although the user model may well be modified as the dialogue progresses). See Wahlster and Kobsa [1986] for an overview of the approaches to these models that have been explored in the literature.

The purpose of a dialogue model is to keep track of the progress of a dialogue. A dialogue model includes a record of the utterances used, and of the plans and goals that have been inferred to explain those utterances. Most of the plan recognition systems for dialogue processing described above contain such a dialogue model either explicitly or implicitly.

Grosz added the notion of *focus* or *attentional state* to the dialogue model [1977, 1978, Grosz and Sidner 1985]. It is used to track the objects and concepts that are most salient at any particular point in a dialogue. Maintenance of an attentional state allows a system to make certain inferences, such as the resolution of pronominal references or of definite noun phrases, that cannot be handled properly by traditional means. Consider for example the following exchange (from Grosz [1977]):

> **Person 1**: The lid is attached to the container with four 1/4-inch bolts.
> **Person 2**: Where are the bolts?

The definite noun phrase 'the bolts' in the second utterance clearly refers to the bolts mentioned by person 1. The use of a focus mechanism allows this relationship to be inferred easily during the analysis of the second utterance. I have largely ignored the attentional component of the dialogue model in my research; however, it would be a useful addition to **PAGAN**.

Carberry [1986b] addresses the general problem of system recovery when a discrepancy arises between the system's dialogue model and the user's dialogue model. She suggests a four-stage approach. First, the discrepancy must be detected. Once it has been detected, the system forms a hypothesis as to the cause of the disparity. Based on this hypothesis, the system interacts with the user in order to determine the actual cause of the disparity. Finally, the system must revise its dialogue model in light of what it has learned through this process.

The other type of model that is of great importance to a plan recognition system is a user model. The purpose of a user model is to keep track of the knowledge and beliefs held by the user. Schuster and Finin [1983] stressed the need for a user model in formulating good responses to questions. User models are also important in understanding the questions themselves. Chin's **KNOME** system [1988] and Nessen's **SC-UM** system [1987] provide good examples of a typical approach to user modeling. In this approach, a set of stereotypes is used to represent typical users. The model of an individual user is composed of that user's stereotype, together with individual facts about that user added on top of the stereotype. For example, **KNOME** and **SC-UM** each utilize four user stereotypes: Novice, Beginner, Intermediate, and Expert.

UC's user model is called **KNOME** [Chin 1988]. In addition to using stereotypical user classes, **KNOME** also stereotypes UNIX commands into the categories Simple, Mundane, Complex, and Esoteric. By combining these two types of stereotype, **KNOME** is able to make a wide range of inferences about the knowledge state of a user based on relatively little information.

Chapter 3

A Theory of Explanation

In its broadest interpretation, an *explanation* of an event is a set of conditions that allow or cause the event to occur. There are a huge number of such explanatory conditions for any given event. To be useful, an explanation must contain only a subset of these conditions. Some of the more important types of conditions are:

1. States of the world
2. Causal relationships
3. Beliefs, intentions, and affect of agents
4. Other events, including actions
5. Physical laws

The types of conditions selected to be part of an explanation dictate the character of the explanation. For example, the following are all potential explanations for the breaking of a particular window:

1. A brick broke it
2. Melissa threw a brick at it
3. Melissa was angry
4. It was made of glass

At first, this final explanation seems to be less an *explanation* of the event than an enabling condition of it. However, it seems to be a perfectly good explanation of the event relative to the question 'How could it have broken?' Other types of explanations include scientific explanations, political explanations, affective explanations, etc. Schank [1986] even suggests explanations composed of questions.

These examples demonstrate that there are many different types of explanations for a single event that might be useful for a particular task. This chapter details what it means for an explanation of an utterance to be a good one. I formalize the concept of 'goodness' by suggesting three criteria by which the quality of an explanation of an utterance may be judged. While these criteria are intended to apply to explanations of

utterances, they are also useful for a broad range of explanation-based tasks.

There are three reasons that it is useful to study criteria for evaluating explanations. First, they can be of help in the design of a knowledge base, because they indicate what types of representations will lead to the desired explanations. Secondly, they assist in the design of a goal analysis algorithm by placing constraints on the output of such an algorithm. Finally, such criteria provide a means to evaluate the output of a goal analyzer, once it has been implemented.

3.1. Criteria For Evaluating Explanations

A good explanation of an utterance meets the following criteria:

1. Applicability
2. Grounding
3. Completeness

The applicability criterion states that a good explanation of an utterance is applicable to the needs of the system that will use that explanation. The grounding criterion states that a good explanation of an utterance is grounded in what is already known of the speaker and of the dialogue. The completeness criterion states that a good explanation of an utterance covers every aspect of the utterance in depth; it leaves no portion of the utterance unexplained. These criteria are valid independent of the algorithm used to construct the explanation. That is, they are criteria that one applies to an explanation to see how good it is, not to an algorithm to see how well the algorithm creates explanations. In the following sections, I describe each of these criteria in more detail.

3.2. The Principle Of Applicability

The principle of *applicability* states that a good explanation of an utterance is applicable to the needs of the system that will use it. That is, no matter how good an explanation might be in other respects, if it doesn't give the system that will use it what that system needs to do its job, then it is not a good explanation.

An assumption that underlies the principle of applicability is that a system that is trying to explain an utterance has interests of its own, which understanding the utterance may help to further. Thus, this principle implies that the operation of a goal analyzer cannot be independent of the system in which it is embedded, and therefore the concept of a domain-independent goal analyzer is impractical. Of course, it may be possible to isolate the portions of a goal analyzer that rely on the particular task to be performed, and represent those portions declaratively. The remaining inference engine would be domain-independent, but the goal analyzer as a whole would not.

The reader may have noticed that I have not included accuracy among the criteria for evaluating explanations. There are two reasons for this. First, accuracy is in general subsumed by applicability. That is, in most cases an explanation of an utterance that does not accurately reflect the speaker's intentions will not be applicable to the needs of the system. Secondly, in some cases, inaccurate explanations are perfectly acceptable. A system designed to simulate a paranoid schizophrenic for example might prefer a delusional interpretation of an utterance to an accurate one.

There are three dimensions of an explanation along which applicability may be assessed:

1. Composition
2. Content
3. Granularity

Applicability of composition concerns the *type* of element out of which explanations should be constructed. Applicability of content deals with the particular choice of elements that compose an explanation. Finally, applicability of granularity covers the level of generality of the elements of a particular explanation. These dimensions are discussed in the following sections.

3.2.1. Applicability of Composition

The principle of *applicability of composition* holds that the *type* of the elements that compose an explanation must be applicable to the needs of the system. The composition of a good explanation is largely a reflection of how that explanation will be put to use; different tasks require different types of explanations. For example, a system whose task is to build a model of a user's knowledge will need to build explanations that are composed of facts about the user that allowed the user to produce an utterance. In such a system, an explanation of:

User: Can you tell me how to delete a file?

might be composed of facts such as:

1. The user does not know how to delete a file.
2. The user believes that the system knows how to delete a file.
3. The user believes that the system will cooperate with the plan.
4. The user knows what a file is, and what it means to delete one.
5. The user understands the task of the system.
6. The user understands English.

For a consulting system such as UC, the knowledge that is applicable to the system's task is knowledge of the plans and goals of the user. This is because the purpose of such a system is to address the user's goals. Thus, an explanation of this question for UC's purposes might be composed of facts such as:

1. The user wants to know how to delete a file.
2. The user expects to be told how to delete a file.
3. The user wants to delete a file.
4. The user wants to free up disk space.

Notice that these facts include both goals that the speaker holds and actions that the speaker expects the hearer to carry out. Because the understanding of a speaker's plans and goals is crucial to so many domains that might benefit from a natural-language interface, my research focuses almost exclusively on making inferences about plans and goals. Occasionally I will mention other types of inferences, usually to place plan and goal analysis in a broader perspective.

3.2.2. Applicability of Content

The principle of *applicability of content* holds that each of the components of a particular explanation should be applicable to the needs of the system. A typical utterance is rarely aimed at achieving a single isolated goal, but rather sits at the tip of a whole chain of motivating goals. Not all such goals will be applicable to a given system though. A good explanation of an utterance will not include all of these goals, but will include only those that are applicable to the purposes of the system. For example, the novice who asks:

User: How can I print a file on the laser printer?

may have asked the question so as to find out how to get a printout, so as to obtain a printout of the file, so as to check its contents for accuracy, so as to turn in an accurate report, so as to get a good grade, so as to graduate with a high grade point average, so as to get a good job, and so on. Only a portion of this chain of motivating goals is likely to be applicable to the purposes of a particular system, and a good explanation will include only that portion. UC's contract with its users for example is to provide information on the use of the UNIX operating system. Thus, in the UC context, a good explanation will not include goals that go beyond the use of UNIX. In this example, the last few goals in the list of motivating goals are beyond UC's purview, and therefore should not be a part of an explanation designed for use by UC.

At the other end of the spectrum, a good explanation of an utterance must include at least one goal that is within the system's domain of expertise. Thus, if an explanation of the utterance:

User: I have a problem with ls.

includes only the goal of informing the hearer of the problem or the goal of the hearer knowing about the problem, that explanation is an inadequate one for the UC domain. An adequate explanation of this statement for UC's purposes would indicate that the user is attempting to solicit help from UC, since this is a goal that is applicable to UC's task. In

practice, adherence to the principle of completeness (described below) ensures that an explanation will have *at least* this desired depth.

3.2.3. Applicability of Granularity

The third dimension along which applicability can be assessed is the granularity of the explanation. The principle of *applicability of granularity* holds that a good explanation contains the right amount of detail. It is often possible to break down a single action into a number of subactions. The extent to which an explanation is broken down in this way is its level of granularity. Different levels of granularity may be more or less applicable to a given system. For example, suppose a user tells UC:

User: I want to add public read permission to my file called jackstraw.

An explanation of this statement that indicates that speaker probably wants to use a UNIX command has a reasonable level of granularity for UC. On the other hand, an explanation that states that the speaker probably wants to type each letter of the name of a UNIX command and each letter of the file name is not a reasonable one for UC's purposes. While it is quite likely to be correct, this explanation is nevertheless too fine-grained to be of use.

Of course, if UC is concerned that the user may not know how to spell the command name, then the latter explanation above may be quite applicable. This points out that the type of explanation that is applicable can change as the short-term goals of the system change. Thus, such goals must always be taken into account when judging the merits of an explanation.

3.3. The Principle of Grounding

The second criterion for evaluating an explanation is the principle of *grounding*. This principle states that a good explanation of an utterance relates what is inferred from that utterance to what is already known about the speaker and the dialogue. Typically, before hearing an utterance, a system will already have some knowledge that is applicable to the processing of that utterance. This knowledge is of two types:

1. Knowledge of the dialogue
2. Knowledge of the speaker

First, the system may have already exchanged some dialogue with the speaker. Secondly, the system may have some stored knowledge about the user, or be able to make informed guesses about what the speaker knows or believes.

It follows from the existence of these two kinds of knowledge that there are two ways that an explanation can be grounded in existing knowledge:

1. By relating it to knowledge of the dialogue
2. By relating it to knowledge of the speaker

The first way that an explanation can be grounded in existing knowledge is by relating it to knowledge of the dialogue. For example, it is difficult to imagine a good explanation for the utterance:

User: It's called crazy-fingers.

that does not relate the utterance to some preceding dialogue. The main reason for this is that the subject of the sentence is a pronoun with no clear referent. Consider now the exchange:

UC: What's the name of the file?
User: It's called crazy-fingers.

In this example, UC's question provides a background against which the user's statement can be understood. A good explanation of the user's utterance must relate the utterance to this previous dialogue.

A second way that an explanation can be grounded in existing knowledge is by lending credence to, or by casting doubt on, something that the system already believes of the user. In this example, the user's response in the exchange:

UC: What's the name of the file?
User: It's called crazy-fingers.

might lend credence to UC's belief that the user knows the 'ls' command (which lists file names in UNIX), thereby grounding the statement in knowledge that UC previously held about the speaker (in addition to grounding it in UC's knowledge of the dialogue). This kind of grounding lies in the domain of user modeling, and I will discuss it only briefly in Chapter 7.

An assumption that motivates the principle of grounding is that the utterance to be explained is part of a dialogue. The significance of this assumption is that systems that don't engage in dialogue, such as simple question-answering systems, can largely ignore the principle of grounding. However, such systems are limited in their usefulness as natural language systems; dialogue is an important part of language.

There are two important aspects of an explanation that is grounded in knowledge of preceding dialogue:

1. The type of concept to which the explanation is attached
2. The type of attachment

First, it is useful to categorize the type of knowledge to which the utterance is connected. There are many such concepts; I am concerned only with plans and goals. An explanation may attach to either of these components of the dialogue model, or possibly to both of them. Secondly, the type of attachment to a concept is important, and can be categorized. The following sections are divided according to the type of knowledge to which an explanation is connected. Within each section, I discuss the ways that an explanation can be attached to a concept of that type.

3.3.1. How an Explanation Attaches to a Plan

There are three ways that an utterance can connect to a plan that a speaker is already pursuing:

1. Plan continuation
2. Plan delay
3. Plan rejection

These types of grounding reflect the status of an existing plan relative to the event to be explained. Whenever there is such an active plan, each new utterance to be explained will relate to that plan in one of these three ways. As there is usually one or more active plans at any point in a dialogue, this aspect of grounding is widespread. The following sections describe each of these types of grounding in detail.

3.3.1.1. Plan Continuation

A good explanation of an event includes every plan in which the event is a step. If such a plan is part of the system's previous knowledge, then the explanation is thereby grounded in previous knowledge. For example, consider the exchange:

UC: Is the file in your directory?
User: Yes.

UC has initiated the plan of first asking a question then receiving a response, so as to achieve the goal of knowing whether the file under discussion is in the user's directory. This plan constitutes a stereotypical sequence of events for which the second step, a yes/no response by the user, has not yet been seen. Therefore, UC's question has set up a specific expectation about how the user will proceed, namely that she will provide a yes/no answer. A good explanation of the user's utterance in this example will include UC's plan, because the uttering of an affirmative response is a continuation of that plan.

It is important to note that this type of grounding is applicable to any goal that has been inferred to help explain an utterance, not just the lowest-level one. For instance, if the user had instead responded to the above question:

UC: Is the file in your directory?
User: Is the Pope Catholic?

the expectation of a yes/no response could not be used directly to ground the explanation that the speaker wants to ask a rhetorical question about the Pope's religion. However, asking such a question is a plan for the goal of conveying an affirmative response. This goal can be grounded in the expectation of a yes/no response by plan continuation. The explanation that the speaker wants to ask an rhetorical question is therefore *indirectly* grounded in the expectation. Thus, a good explanation of the user's utterance in this example would indicate that the question was asked as a plan for the goal of conveying an affirmative answer, and that conveying an affirmative answer was done as a plan for the adopted goal of UC knowing the answer to its question.

In dialogue, plan continuation may be signaled by a participant before it occurs:

UC: What is the name of the file?
User: Hold on, let me check.

Here, the user has initiated a plan of informing UC about an upcoming plan continuation. The remaining steps of the user's plan are to obtain the requested information, then to inform UC. This plan as a whole constitutes a continuation of UC's plan.

3.3.1.2. Plan Delay

When one participant in a dialogue (or in fact in any cooperative venture) initiates a plan that requires the participation of other agents, it does not follow that the other participants will readily accept the plan and agree to continue it. They may instead decide that for some reason they should take another direction. There are two general categories into which phenomena of this type may be placed: plan delay, and plan rejection. Plan delay, an interruption in the execution of a plan, is the subject of this section. Plan rejection, a termination of the execution of a plan by one of its participants, is discussed in the next section.

Plan delay occurs when a plan participant wishes to delay the execution of the plan until some later time. There are three types of plan delay:

1. Clarification delay
2. Skepticism delay
3. Extra-schematic delay

A clarification delay is a delay designed to fill a gap in an agent's knowledge of the plan or goal. A skepticism delay is one in which the agent fully understands the plan or goal, but questions its validity. Finally, an extra-schematic delay is one that arises due to circumstances outside of the plan being delayed. These three types of plan delay are the subject of the following sections.

3.3.1.2.1. Clarification Delay

Clarification delay is delay designed to fill a gap in a participant's knowledge of the plan:

> UC: What is the name of the file?
> User: Do you mean the file I want to copy?

In this example, the user is not sure about what question is being asked. Rather than selecting a particular interpretation of the question and answering it, thereby continuing the plan, the user decides to get clarification about the meaning of the question from UC. This delays UC's plan until the clarification is made.

Clarification delay is terminated (that is, the original plan continues) when the subplan is completed via plan continuation (see above), or when it is discarded via plan rejection (see below). In this example, once UC indicates the intended meaning of the question, the user can continue UC's original plan.

3.3.1.2.2. Skepticism Delay

Skepticism delay arises when a plan participant fully understands the plan being delayed, but is skeptical about its efficacy or efficiency:

> UC: What is the name of the file?
> User: I may not have it.

Here, the user is questioning whether UC's plan will work at all. The user can also suggest that there might be a more efficient plan:

> UC: What is the name of the file?
> User: Can't you look for it?

In this example, the user is delaying UC's plan by questioning whether it wouldn't be more efficient for UC to search out the information itself.

Skepticism delay is completed, and the delayed plan is continued, when the skepticism is allayed. Since it is possible that the skepticism might not be allayed, the system must have the ability to infer that plan delay has been converted to plan rejection. Plan rejection is discussed below.

3.3.1.2.3. Extra-Schematic Delay

The third kind of plan delay is extra-schematic delay. Extra-schematic delay is plan delay that arises due to concerns outside of the plan being delayed:

> UC: What is the name of the file?
> User: That reminds me of the one about the file and the Stanford student...

In this example the plan delay is triggered by a component of UC's plan, but the plan introduced by the user does not itself bear on UC's plan.

3.3.1.3. Plan Rejection

The other way that an utterance may fail to continue an existing plan is by rejecting the plan as flawed in some way. As with skepticism delay, plan rejection may stem either from efficacy or efficiency considerations. A plan participant may reject a plan for reasons of efficacy either because the plan cannot be carried out, or because one of the effects of the plan is undesired. For example, a participant may be unable to answer a question:

> UC: What is the name of the file?
> User: I don't know.

Here, UC has asked the user for the name of some file under discussion. UC now expects the user to tell it the name. Since the user doesn't know what the name is, she rejects the plan by indicating her inability to provide the answer.

A plan participant may reject a plan for reasons of efficiency either because there is a better way to achieve the desired outcome, or because the execution of the selected plan would be too expensive in an absolute sense. For example, the participant may indicate that another agent would be better able to perform a portion of the plan:

> UC: What is the name of the file?
> User: You should ask Marcia.

Here, a new plan is suggested by the user, thereby rejecting UC's initial plan.

3.3.1.4. The Relationship between Plan Delay and Plan Rejection

Although it is useful to treat plan delay and plan rejection as completely separate phenomena, they are in fact closely related. This is because there is never a guarantee that a delayed plan will be resumed. Thus, when the system detects plan delay, it must have the ability to treat it as plan rejection. Furthermore, plan rejection will never be absolute, because the speaker always has the option of taking it up again. Thus the distinction between plan delay and plan rejection is one of degree.

3.3.2. Connecting an Explanation to a Goal

There are three ways that an event may attach to a goal. These ways parallel the ways that an event can attach to a plan:

1. Goal achievement
2. Goal scrutiny
3. Goal rejection

These three types of attachment to a goal are described in the following sections.

3.3.2.1. Goal Achievement

An action can achieve a goal, either by continuing an existing plan, or by introducing a new plan for an existing goal [cf. Wilensky 1983]. The former case occurs when an action is grounded by plan continuation. The latter case occurs when an action does not conform to a specific expectation, and instead addresses a higher-level goal (that is, a goal that motivated the plan containing the expected action). Suppose that instead of responding "yes," as in an earlier example, the user engages in the following dialogue:

UC: Is the file in your directory?
User: It's in /tmp.

Many accounts of yes/no questions have tried to view responses such as this one as an answer to the yes/no question asked. Such approaches require a refinement of the notion of what it means to be a yes/no question. For example, Hirschberg [1984] represents yes/no questions as scalar queries rather than as questions that take simple yes/no answers.

In contrast to such positions, I believe it is important to view yes/no questions themselves as questions that can only take yes/no answers or answers that can be inferred to indicate yes/no answers *independent of the context of the question* (as in the response 'Is the Pope Catholic?'). The real complexity inherent in yes/no questions arises because yes/no questions are typically used in service of higher-level goals. A *response* to a yes/no question can address such a higher-level goal, but an *answer* to a yes/no question

can only be yes or no.

The above example is a case in point. There is no way to construe this response as either an affirmative answer or a negative answer to the question.[1] Thus, the lowest-level expectation is not met. However, the goal UC was addressing in asking the question was to know the location of the file. This goal is itself an expectation, because it is a step of a higher-level plan that has not yet been completed. By informing UC of the file's location, the response succeeds in addressing this goal, independent of UC's original plan of receiving a yes/no response to its question; the utterance therefore matches the expectation that the user will address the goal. Thus, an utterance such as this one is grounded both by goal achievement, and by plan rejection.

In general, it is not acceptable simply to respond negatively to a question such as this one:

> UC: Is the file in your directory?
> User: No.

This is because such a question is usually in service of the goal of knowing the location of the file, and whereas an affirmative response satisfies this higher-level goal, a negative response does not. However, there are cases where a negative response is perfectly appropriate. For example, there are some UNIX utilities that require that certain files are located in the user's directory. If the user were inquiring as to why such a utility wasn't working, then the above exchange is perfectly reasonable. The point here is that it is not a goal analyzer's responsibility to determine whether a response such as this one is malformed. Rather, this task should fall to a separate component that detects user misconceptions. See Pollack [1984], Chin [1988], and Quilici [to appear] for examples of such systems.

3.3.2.2. Goal Scrutiny

The second way that an event can be grounded in a goal is by questioning that goal in some way. Consider for example the exchange:

> UC: What is the name of the file?
> User: Why do you want to know?

Here, the user is skeptical of UC's need for the information it is requesting, and so questions whether UC's goal is a legitimate one. Notice that in addition to questioning UC's goal, this response also delays UC's plan.

---

[1] This response *implies* a negative answer, but only by virtue of the analysis described here.

3.3.2.3. Goal Rejection

The third way that an event can be grounded in a goal is by rejecting that goal. This may occur either because the goal should not be achieved, or because it cannot be achieved. For example, in the following exchange the user indicates that there is a reason that UC's goal should not be fulfilled:

> UC: What is the name of the file?
> User: I promised Sharon I wouldn't let anyone find out.

Thus, UC's plan is rejected.

The other reason that a participant may reject a goal is if the participant believes that the goal cannot be achieved. For example, the goal may not be achievable if some premise of the goal is incorrect. Consider the exchange:

> UC: What is the name of the file?
> User: I haven't typed the data into a file yet.

Here, UC has been led to believe that the data are contained in a UNIX file. This premise is incorrect, and consequently it is impossible for the user to answer the question. Instead, the user indicates the reason that the question is flawed. The extreme case of this phenomenon is when the user's overall goal has been incorrectly analyzed:

> UC: What's the name of the file you want to delete?
> User: I don't want to delete anything.

Here, UC has incorrectly inferred that the user desires to delete a file, and has initiated a plan to determine which file is to be deleted. Since the user never intended that a file be deleted, she rejects UC's plan.

3.4. The Principle of Completeness

The principle of *completeness* states that a good explanation of an utterance covers every aspect of the utterance; it leaves no portion of the utterance, or of the explanation itself, unexplained. There are two kinds of completeness:

1. Depth Completeness
2. Breadth Completeness

Depth completeness (also called *vertical* completeness) is completeness of a single line of explanation of an utterance. An explanation of an utterance exhibiting depth completeness includes a goal that motivated the production of the utterance, and a goal or theme to explain each inferred goal. Breadth completeness (also called *horizontal* completeness) is coverage of all aspects of the utterance. An explanation that exhibits

breadth completeness includes every goal that motivated the production of the utterance. The following sections describe these two kinds of completeness.

3.4.1. Depth Completeness

When an explanation is inferred for an utterance, that explanation itself may be subject to explanation. For example, the goal of requesting that UC tell the user how to compress a file might explain the utterance:

User: Can you tell me how to compress a file?

That goal itself can be explained by the user's goal of knowing how to compress a file, which might in turn be explained by the user's goal of compressing a file, etc. The depth completeness criterion states that a good explanation includes an explanation of every goal inferred in this way.

Of course, if the only type of explanation for a goal were another goal, strict adherence to the vertical completeness criterion would require infinite explanations. There are two ways around this apparent quandary. First, a non-intentional explanation can be used to explain a goal, thereby terminating the chain of explanation. Secondly, the principle of applicability can be called into play. The principle of completeness is often at odds with the principle of applicability. Where the applicability criterion dictates that a particular portion of an explanation should be omitted, the completeness criterion dictates that it should remain a part of the explanation. In such cases, the applicability criterion takes precedence; there is little sense in having a complete explanation of an utterance if it isn't applicable to the task at hand. The *modified* depth completeness criterion is then that a good explanation includes an explanation of every inferred goal *in the domain of discourse*.

3.4.2. Breadth Completeness

The breadth completeness criterion requires that a good explanation of an utterance or of an inferred goal include every motivating goal of that utterance or inferred goal. It is derivative of Wilensky's notion of exhaustion [1983]. The breadth completeness criterion is based on a general principle of planning, namely that it is often possible to address more than one goal with a single utterance. For example, a person can both ask what vi does, and indicate a suspicion that it is an editor, with a question like:

User: Is vi an editor?

The most obvious way to address two goals with a single utterance is by explicitly listing two requests, as in:

User: I want to know how to rename a file and how to copy a file.

One might claim that such a statement is no more than a weak cohabitation of two separate statements within the same sentence. However, until a principled way is available to determine how a given input should be chunked before it is processed, sentences will remain the natural delimiter. Thus, an utterance such as this one must be handled as a single request, since it uses a single sentence. A complete explanation of this statement must then include both the goal of knowing how to rename a file, and the goal of knowing how to copy a file.

The incorporation of multiple goals into a single utterance need not be so straightforward. For example, a single utterance may be used both to request something of the hearer, and to inform the hearer of a particular fact. The question:

User: How do I delete the file 'binkle' in my top-level directory?

both requests that the hearer say how to delete the file, and informs the hearer of the whereabouts of the file. A good explanation of this question must include both of these as goals of the speaker.

It is possible to use a single utterance to address virtually any pair of goals that can themselves be addressed linguistically. Thus a taxonomy of pairs of goals that might be addressed by a single utterance is no easier to come by than a taxonomy of goals themselves. Consequently, I will not expound on such a taxonomy beyond providing a number of examples. The interested reader is referred to Appelt's work [1985] for a description of this problem from a generation perspective.

To see the importance of recognizing multiple goals, consider this example:

User: I want to delete a directory and all the files in it.

The user here has expressed two goals explicitly in a single utterance. To successfully respond to this utterance, the system must recognize *both* goals. If the system recognizes only the goal of deleting the directory, it may come up with the plan of moving its files elsewhere and then deleting it; if it recognizes only the goal of deleting the files, the plan it chooses will almost certainly leave the directory intact. Thus, it is important for **PAGAN** to infer *all* the goals that motivate the production of an utterance.

Consider another example:

User: How can I prevent anyone from reading my file without deleting it?

The user has two goals in making this utterance: to prevent other people from reading the file, and to preserve the file. Once again, the goals are distinct and non-conflicting, and both must be addressed to adequately respond to the question. The first goal is the user's main goal; the second is called an *adjunct goal* [Wilensky 1983]. However, the method for inferring these goals, given a representation of the utterance, is less straightforward

than in the previous example, since they are not mentioned as a statement of goals but rather as a question.

Addressing multiple goals with a single action is not limited to goals in the domain of discourse. Discourse goals themselves may be pursued in this manner:

User: My last question is how to save my file once I've edited it.

The speaker of this utterance has the domain goal of finding out about how to save a file while in the editor.[2] She also has the discourse goal of indicating that she expects the dialogue to terminate after this problem has been resolved. While awareness of this latter goal may not be useful to a system that simply performs rote question answering, it will be helpful to a more advanced system in planning its own utterances.

Finally, consider the statement:

User: I have a problem with ls.

The user who initiates a conversation with UC by saying this probably has several goals in mind, including to initiate a dialogue with UC, to solicit help from UC, and to inform UC that 'ls' is the topic of the problem. In many systems, goals such as these are not explicitly inferred. Rather, the system embodies assumptions that its authors have made about the types of higher-level goals that the system's users will have. Encoding this type of knowledge procedurally is not necessarily bad. However, the system should have the ability to represent and treat this knowledge declaratively should the need to do so arise.

The validity of the horizontal completeness criterion is based on the assumption that the interests of the system that will use the explanation are not so constrained as to be single-minded. If the system *is* single-minded about its goals, then this criterion isn't useful. For example, a simplistic database lookup program will generally only need to extract a database query from the user's utterance. It won't matter to such a program whether a full understanding of an utterance is achieved, only that a database query is selected. For systems with broader applicability though, this is an important criterion.

3.5. Summary

This chapter introduced three criteria for good explanations of utterances. The first, the principle of *applicability*, states that a good explanation of an utterance is one that is applicable to the needs of the system that will use that explanation. Thus, an explanation is only a good one relative to some intended purpose. The second criterion, the principle of *grounding*, states that a good explanation of an utterance relates what is

[2] Of course, the speaker also has the discourse goal of asking how to save the file. However, this goal is subservient to the domain goal. I am concerned in this section only with multiple goals that are not causally related to one another.

inferred from that utterance to what is already known about the speaker and the dialogue. The last criterion, the principle of *completeness*, states that a good explanation of an utterance covers every aspect of the utterance. The principle of completeness is further broken down into *depth* completeness and *breadth* completeness. The depth completeness criterion dictates that each component of a good explanation must itself be explained. The breadth completeness criterion dictates that each component of the utterance to be explained must be covered by the explanation. Together, these three criteria provide a means to determine the quality of an explanation of an utterance.

Chapter 4

Representation of Plans and Goals

Since the vocabulary of goal analysis is plans and goals, any approach to goal analysis must include a means to represent knowledge about plans and goals. Most approaches use a hybrid of two representations:

1. STRIPS operators
2. Scripts

These traditional representations have two main problems that limit their usefulness in goal analysis:

1. The intended effect problem
2. The precondition problem

The intended effect problem arises when a representation fails to distinguish adequately the intended effect, or goal, of a plan from other outcomes of the plan. The precondition problem arises when preconditions are used to represent more than one kind of knowledge. In this chapter, I first describe STRIPS operators and scripts. For each of these representations, I demonstrate how the intended effect problem and the precondition problem arise. I then introduce a new representation, called a *planfor* [cf. Wilensky 1983]. This representation combines the good aspects of the script and operator approaches, while at the same time solving the intended effect problem and the precondition problem. Finally, I discuss some processing advantages provided by planfors.

4.1. STRIPS Operators and Their Descendants

One of the earliest successes in mechanized planning was the STRIPS program [Fikes et al. 1971]. STRIPS was a planner that constructed plans that a robot could use to solve simple movement problems. For example, STRIPS might produce a plan for bringing a box into a room from an adjacent room, that consisted of first moving into the

adjacent room, then pushing the box into the original room.

STRIPS' knowledge of planning was stored as *operators*. A STRIPS operator is a description of how a hypothetical action affects a world model. The world model is a collection of first-order predicate calculus well-formed formulae (WFF's). Each operator is composed of a header (the name of the operator together with a list of parameters), a set of preconditions (conditions that must hold in the world model for the operator to be applicable), and a description, in the form of WFF's to be added to and deleted from the world model, of the way in which that operator changes the world model.

```
OPEN(dx)  ; open door dx
    preconditions:
        TYPE(dx,door),
        STATUS(dx,closed),
        NEXTTO(robot,dx).
    deletions:
        STATUS(dx,closed)
    additions:
        STATUS(dx,open)
```

Figure 4.1. The STRIPS OPEN Operator

An example of a STRIPS operator is shown in Figure 4.1. This operator represents the knowledge STRIPS needs to construct plans that include the opening of doors. The preconditions to the operator are that the object in question is a door, that the door is closed, and that the robot is next to the door. If the operator is executed, it will change the model of the world by deleting the fact that the door is closed, and adding the fact that the door is now open.

```
BOARD(agent, train, station)
    applicability conditions: SOURCE(train, station),
      DEPART.LOC(train, loc), DEPART.TIME(train, time)
    precondition: AT(agent, loc, time)
    effect: ONBOARD(agent, train)
```

Figure 4.2. The BOARD Plan

More recently, STRIPS operators have been generalized for use in plan recognition. As a representative example of this approach, I will use the BOARD plan described by Allen and Perrault [1980]. The BOARD plan is shown in Figure 4.2. This plan representation says that BOARD is a plan with three parameters: an agent to do the boarding, a train to be boarded, and a station in which the boarding is to be accomplished. The

applicability conditions are simply further specifications of the parameters; they indicate that the train is originating from the station, and that it is leaving from some unspecified location at some unspecified time. The precondition states that in order to perform a **BOARD**, the agent must be **AT** the unspecified location at the unspecified time. The effect of performing the **BOARD** is that the agent is then **ONBOARD** the train.

4.2. Scripts

Early work in story understanding made use of a representation called a *script* [Schank and Abelson, 1977]. A script is a "standard event sequence" [p. 38]. That is, a script is used to group events that typically occur in fixed (or semi-fixed) sequences. For example, a restaurant script might include steps for entering the restaurant, ordering food, being served, eating, paying, and leaving the restaurant. An abridged version of the restaurant script [p. 43] is shown in Figure 4.3.

$RESTAURANT
entry conditions:
customer is hungry
customer has money
results:
customer has less money
owner has more money
customer is not hungry
customer is pleased (optional)
scene 1: entering
scene 2: ordering
scene 3: eating
- waiter gives food to customer
- customer ingests food

scene 4: exiting

Figure 4.3. The Restaurant Script

Scripts were developed as a way to represent the knowledge needed to understand simple stories. They are useful for making various inferences about the actions taken by the characters in the story. Once such an action matches a portion of a script, the remainder of the script can be used to predict future events. This allows events that continue the script to be processed rapidly. For example, upon learning that Belinda entered a restaurant, the restaurant script might be instantiated, and the observed event matched with the first scene of the script. The remaining steps in the script could then be used to facilitate the explanation of subsequent events, such as Belinda ordering a grilled tomato and cheese sandwich, the waiter bringing it, and Belinda paying.

Scripts can also be used to infer that events that were not observed actually took place. In this example, the system might infer that Belinda ate the sandwich, and that she left the restaurant. Of course, such events may not have taken place at all. However, because scripts represent typical events, it is likely that under normal circumstances they did occur.

An important feature of the script representation is that it can group actions taken by multiple agents into a single framework. Notice that the restaurant script includes both actions taken by the customer (e.g. customer ingests food), and actions taken by other important agents (e.g. waiter gives food to customer). Incorporating knowledge about multiple agents into a plan representation allows planners and understanders that use that knowledge to make inferences about the actions of other agents without recourse to complicated analysis of mutual belief. In this example, the system that is using the $RESTAURANT script to explain Belinda's actions does not need to determine what beliefs the waiter holds that influence him to bring the food; it may assume that the waiter will bring the food because that is typically what happens in this situation. In summary then, scripts provide a good way to associate related actions that are performed by multiple agents, so as to allow useful inferences about those actions.

4.3. The Intended Effect Problem

Every plan must have an associated goal. It makes little sense to say that a person has a plan, but that it is not a plan *for* anything. A major problem with these two families of plan representations is that the relationship between the plan and the goal the plan is intended to accomplish is unclear. A plan representation should include an explicit representation of the goal that normally underlies the use of that plan. The inclusion of the normal underlying goals in a plan representation constrains the plan recognition search process, which would otherwise have to examine all the possible effects of an inferred plan. I call the problem of the lack of an explicit representation of the goal of a plan the *intended effect* problem.

Consider first the STRIPS operator representation. While they are useful in planning situations, from an understanding perspective, STRIPS-style operators are not very useful. The main reason that STRIPS operators are not useful for understanding is that the logical place to represent the goal of a plan, the effects slot, is allowed to contain multiple effects. When an action is observed, and the operator that corresponds to that action is inferred as an explanation of the action, it remains unclear which of these effects were intended, and which were merely side-effects. Thus, STRIPS operators represent *actions* rather than *plans*. For example, the BOARD plan might have another effect such as NOT-AT(agent,loc,time2), indicating that a person who boards a train is no longer on the platform. If so, the BOARD plan would not be helpful for determining whether a person who boarded a train did so in order to be on board the train, or in order to no longer be on the platform. While an additional effect such as this might not be a problem for a planner (which might well want to formulate a plan for leaving a train platform), it is a big problem for an understanding system that is trying to determine why a person would board a train.

Scripts too suffer from the intended effect problem, although this an act of omission rather than one of commission. Scripts in Schank's formulation for example do not facilitate inferences about the script as a whole. That is, although scripts are useful for inferring how actions relate to each other, as Wilensky [1978] points out, they are not useful for inferring how the sum of the actions that compose the script relates to other knowledge that the system may have. For example, the **$RESTAURANT** script can be used to explain why Belinda ordered a sandwich (because that was the action called for at that point in the script[1]); without the inclusion of an associated goal in the script however, it cannot be used to explain why she wanted to go to the restaurant in the first place.

4.4. The Precondition Problem

The second main problem with these classes of representations is the *precondition problem.* The notion of preconditions typically conflates two different kinds of knowledge about when a plan is applicable:

1. Definitional knowledge
2. Heuristic planning knowledge

The first kind of knowledge is that of the defining properties of the actions that compose the plan. For example, part of what it *means* for an agent to perform an action at a particular location is that the agent is at that location. Or, in Hendler's words [1986, p. 92], "the set of preconditions for a frame is exactly the frame itself." This kind of precondition is called a *definitional* precondition.

The other type of knowledge that goes into the traditional notion of preconditions is heuristic planning knowledge. Preconditions that express heuristic planning knowledge are called *heuristic* preconditions. Such knowledge says for example that being at the location where an action is to take place is an important property of that action for which planning must often be done. Therefore, a planner should look carefully at this property to make sure it holds, if it wants to use the plan. At the other extreme, the property that an agent be able to move in order to perform a physical action, while certainly a definitional precondition, is nevertheless not a condition with which a planner would want to concern itself in the absence of some explicit indication that movement might be impaired. This is because capability of movement is so likely to hold that there is rarely a case in which failing to consider it will lead to plan failure or plan recognition failure. Thus, such conditions need not be present as heuristic planning knowledge for planning or plan recognition to be effective.

---

[1] Notice that this is a non-intentional explanation. That is, it is an explanation relative to a social convention, not to a personal intention. In omitting intentional information, such explanations are incomplete; they nevertheless may be applicable to a particular task.

In addition to this confusion about the nature of preconditions, two other problems are typically found in explicit representations of preconditions. First, any given action will have many preconditions. For example, to board a train, not only must you be at the gate prior to the time the train departs, but the train must also be there, and its doors must be operational, and you must be able to move, and you must fit through the train doors, and gravity must still work, etc. The vast number of such preconditions makes their explicit representation as preconditions to *each* plan that must use them prohibitive. Secondly, most preconditions are applicable to many plans. The precondition that one be at a particular location is applicable to *any* plan designed to take place in a specified locale, not just the train-boarding plan. A plan representation should take such commonalities into account, and represent the appropriate knowledge in only one place.

4.5. The Planfor Representation

To address these problems, the concept of a *planfor* is useful [cf. Wilensky 1983]. A planfor is a relation between a type of goal and a hypothetical event (called a plan) that constitutes a possible method of achieving a goal of that type. Typically, such plans represent compound events composed of a partially ordered set of hypothetical actions. Planfors unite the idea of a script as a way to group related actions of multiple agents with the notion of a plan that is being pursued by a single actor in order to achieve a particular goal. Planfors highlight the notion that a plan is a way to achieve a goal, as opposed to an abstract structure bearing no direct explicit relation to a goal. They also allow that a planner may intend that another agent perform some action as a part of the plan, without being in a position to control the other agent's performance of the action. The following examples illustrate these points.

A planfor can be used to associate a goal with a plan of any type. For example, any simple indirect speech act of the form:

> Do you know how to *perform task*?

can be associated with a goal by this planfor:

> **Goal:** Speaker ask hearer how to perform task
> **Plan:** Speaker ask hearer whether hearer knows how to perform task

Planfors can also be used to represent knowledge about dialogue. A question and answer sequence aimed at finding out how to perform a particular task can be represented by this planfor:

> **Goal:** Speaker know how to perform task
> **Plan:** Speaker ask hearer how to perform task
> Hearer tell speaker how to perform task

Notice that the plan in this example includes an action on the part of the hearer that is not under the speaker's control. I draw a distinction between a plan, which is an abstract sequence of one or more steps aimed at achieving a particular goal, and the specific sequence of actions that a planner executes in conforming to such a plan. The former may include actions by agents other than the planner. The latter is the planner's implementation of the plan, and therefore includes only actions taken by the planner. Such actions may include waiting for some agent other than the planner to perform an action.

Finally, planfors can be used to represent the relationship between plans and goals in the domain of discourse. For example, knowledge about the UNIX **cp** command can be represented as:

Goal: Have copy of file1 in file2
Plan: Execute **cp** program with arg1 = name of file1, arg2 = name of file2

The use of planfors allows plans of different types to be combined and associated with a single goal. For example, the first two planfors shown above can be combined to express both an indirect speech act and a question and answer sequence:

Goal: Speaker know how to perform task
Plan: Speaker ask hearer whether hearer knows how to perform task
Hearer tell speaker how to perform task

4.5.1. Solution to the Intended Effect Problem

The intended effect problem arises because the representation of plan knowledge expresses facts about *causality*, but the knowledge needed for goal analysis is knowledge of *intention* and *typicality*. Planfors solve this problem by explicitly representing knowledge of typical intention. In fact, planfors do not represent fundamental knowledge of causality at all. There is usually a causal relationship between a plan and a goal that are connected by a planfor. However, the planfor itself does not represent the causality. What the planfor does represent is a notion of typicality. It indicates that its plan is one that is typically or conventionally used to achieve its goal.[2] For example, the UNIX **rm** command may cause a file to be deleted. It may also cause the disk arm to be moved. It would be a mistake though to say that **rm** should be connected to the goal of moving the disk arm by a planfor relation; **rm** is simply not something one would typically use to move the disk arm. On the other hand, **rm** should be connected to the goal of deleting a file by a planfor relation, since this is the goal for which **rm** is typically used. By using planfors as inference rules in plan recognition, the search space for explanations can be greatly restricted.

[2] It may also be the case that a plan is *the* normal plan for its associated goal, but this condition is not a part of the meaning of a planfor.

Planfors do not represent immediate knowledge of actions. Rather, they represent compiled knowledge of the effects that actions typically have. Consequently, planfors do not represent the last word in goal analysis knowledge. If planfor-based goal analysis fails, the system should have the option of retreating to action-based goal analysis. Such analysis will be more costly than planfor analysis, precisely because planfors represent compiled knowledge. For example, it is possible that a user could use **rm** so as to move the disk arm. However, since these two are not connected via a planfor, planfor analysis will not be useful in understanding this scenario. **Rm** is connected to the goal of moving the disk arm by a causal relation though, since **rm** can cause the disk arm to be moved.[3] Such causal knowledge is useful in understanding this scenario. However, in moving from planfor analysis to causality analysis, the system may need to examine *all* the possible effects of using **rm**; doing so can be costly, and consequently planfor analysis is to be preferred over causality analysis.

Another type of processing that may need to be called into play if planfor analysis fails is analysis of mutual belief. It is the embodiment of convention in planfors that allows **PAGAN** to perform goal analysis without requiring knowledge of mutual belief. However, if planfor analysis is not sufficient to understand an utterance, it may be necessary for the system to determine which facts are mutually believed, and to use that information to explain the utterance. Allen [1979] describes methods for representing mutual belief, and using it to explain an utterance.

4.5.2. Solution to the Precondition Problem

Planfors solve the precondition problem by distinguishing between heuristic planning knowledge and knowledge of the defining properties of actions. First, unlike formalisms derivative of **STRIPS** operators, planfors do not explicitly represent heuristic preconditions at all. Rather, an agent such as a planner or a goal analyzer that requires such knowledge has the option of adding it on top of the planfor representation. For example, to delete a UNIX file, it is necessary both that the parent directory is writable, and that the disk arm is operational. A planner might want to have special metaplan knowledge that it should check the protections on the parent directory before settling upon a plan that involves deletion; it probably would not want to attach the same degree of importance to checking the status of the disk arm before using deletion as a plan step. Luria [1987, 1988] presents a planner called **KIP** that makes decisions based on this notion of 'degree of concern.'

Secondly, in the planfor representation, definitional preconditions are represented as constraints on the concepts that make up the plan. This is done by placing each component concept at its correct location in the knowledge hierarchy. The use of a hierarchical formalism that allows inheritance (such as a semantic net) is dictated by these considerations.

---

[3] Such a causal relation is unlikely to be explicitly represented in the knowledge base; rather, it would have to be inferred from more basic knowledge about **rm** and disk arms.

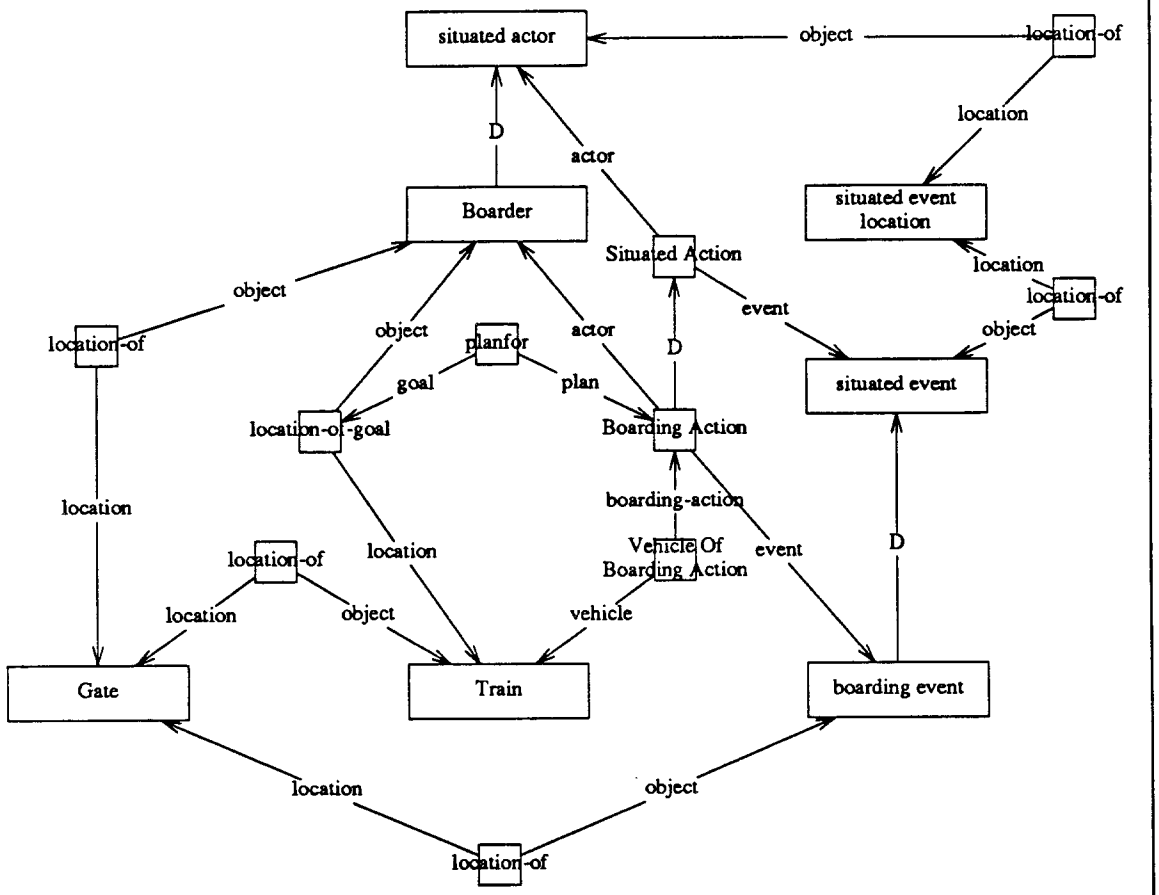

Figure 4.4. Representation of the AT precondition of the BOARD plan

The upper right portion of this diagram shows how a SITUATED-ACTION *is defined so as to restrict the actor to be at the same place (i.e. the* SITUATED-EVENT-LOCATION*) as the action. The remainder of the diagram shows how a boarding action can be defined as a situated action.*

As an example, suppose the BOARD plan were to be expressed in the planfor representation. The AT(agent,loc,time) precondition indicates that the agent of the plan must be at the specified location at the specified time for the plan to be useful. In a planfor, this precondition would be represented by forming a single SITUATED-ACTOR concept. Knowledge about the time and location of the event would be predicated about this new concept. SITUATED-ACTOR would also be used in place of the more general ACTOR concept as the actor of the boarding action. Thus, the time and location of the event would be part of the representation of the event itself, rather than being tacked on as additional preconditions. A picture of the relevant portion of this representation is shown in Figure 4.4. To simplify the diagram, the direct relationship between the train and the boarding event is not shown.

Rather than being attached to individual plans, definitional preconditions are now a part of the knowledge hierarchy at large. This means that the huge number of preconditions, while still a problem, is not a problem for plan representation; rather, the burden of maintaining a large number of facts is placed on the knowledge base as a whole, where it belongs. Furthermore, preconditions that are applicable to more than one plan can be placed above each such plan in the knowledge hierarchy. This allows the information to be placed in a single location, yet be available via inheritance to each plan that needs it.

It would be delightful if it were possible to use the defining properties of actions to automatically infer heuristic preconditions. Unfortunately, there are several obstacles that any such method must overcome. First, the system must have detailed knowledge about which events make up the background knowledge needed for the understanding of each action. Secondly, the system must have causal knowledge about how (or whether) each component of this background contributes to the action in question. Thirdly, the system must have some way to cope with the frame problem that arises when considering the effects of the events it is testing as potential heuristic preconditions. Each of these obstacles is significant; consequently, the best way to maintain knowledge of heuristic preconditions is to express such knowledge explicitly in the knowledge base.

4.5.3. Processing Advantages

In addition to the representational advantages discussed above, planfors also provide processing advantages. Planfors allow **PAGAN** to combine certain inferences that should be kept together:

1. Inferences about plans and goals
2. Inferences about actions and intended responses
3. Inferences about linguistic goals and domain goals

First, inferences about plans may be made at the same time as those about goals. This is in contrast with systems such as Wilensky's **PAM** system [1978] that infer plans and goals separately. When analyzing the sentence "John walked over to Bill," for example [pp. 37-38], **PAM** first infers that the action of walking was used as part of the plan for John's moving under his own power. It then tries to find a reason that he was pursuing that plan, and infers that John has the goal of being near Bill. If a planfor relation were used to represent the knowledge required to make these inferences, both of the inferences could be made simultaneously. Once a planfor has been inferred to explain an action, both the plan and the goal of the plan are immediately available without additional processing. Thus, the use of planfors in plan recognition incorporate Allen's separately-defined Precondition-Action Rule, Body-Action Rule, and Action-Effect Rule (described in Chapter 2) into a single class of inference rule.

Secondly, inferences about plan recognition and inferences about intended response recognition may be combined when using planfors. This is done by including the intended response in the plan and associating this entire plan with a single goal. This contrasts with the operation of systems such as Sidner's [1985] that first perform plan

recognition, then worry about what response was intended. For example, in trying to explain the statement "I want the vase on the table," Sidner's system would first infer that the speaker had the plan of informing the system about the goal of having the vase on the table. It would then go on to make a separate inference that the speaker expects that the system will respond by adopting that goal, and placing the vase on the table. The use of planfors allows both of these inferences to be made concurrently. In this example, a planfor could be used that included a two-step plan: first that the speaker state a goal, then that the hearer respond by satisfying that goal. The ability to perform both kinds of inference simultaneously conforms to the intuition that no extra processing is required to determine for example that an answer is expected once the realization is made that a question has been asked.

Finally, planfors allow inferences about linguistic goals and about domain goals to be handled by a single inference engine. The separation of goal analysis into linguistic goal reasoning and task goal reasoning [cf. Allen, Frisch, and Litman 1982] is unnecessary; the difference between the two lies only in the type of actions that make up the steps of the plan, not the form that those plans take. For example, both the communicative plan of asking a question, and the domain plan of using **rm**, can be represented by planfors. The same processes that explain and disambiguate questions can then be used to explain and disambiguate uses of **rm**.

4.6. Summary

This chapter introduced a representation for knowledge about plans and goals called a planfor. A planfor is a relation between a type of goal and a hypothetical event (called a plan) that constitutes a possible method of achieving a goal of that type. Planfors solve two problems found in other approaches to plan representation. First, planfors explicitly distinguish the intended effect of a plan from other effects of the plan. Secondly, planfors distinguish definitional properties of the events that make up a plan from heuristic planning knowledge about that plan. In addition to solving these problems, planfors allow inferences about plans, about goals, and about expected actions to be made concurrently, as opposed to performing each type of inference individually.

Chapter 5

Finding an Explanation

Recall from Chapter 1 that the goal analysis algorithm is divided into three steps: finding an explanation, handling ambiguity, and extending the explanation. This chapter is devoted to the first step of the algorithm. Chapters 6 and 7 cover the processes of detecting and handling ambiguity. Finally, Chapter 8 discusses the conditions under which an explanation of an utterance is extended.

The first step of the goal analysis algorithm is to find an explanation for the utterance. The method suggested here is a variant of the **PAM** explanation algorithm [Wilensky 1978], adapted for use in dialogue. There are three places that **PAGAN** may find an explanation for an utterance:

1. Among its expectations of speaker actions
2. Among the abstract planfors in its knowledge base
3. Among the non-intentional explanations in its knowledge base

Each of these sources is described in one of the following sections.

5.1. Meeting an Expectation

The first place that **PAGAN** may find an explanation for an utterance is among its expectations of actions to be taken by the speaker. **PAGAN** maintains a representation of the structure of the dialogue that has transpired. This model is composed of plans and goals, along with the status of each plan. It may contain plans that have not yet been completed. When an action to be explained matches a pending step of a partially executed plan, the goal associated with that plan constitutes a possible explanation of the action. The process of comparing an utterance to pending plan steps is called *expectation matching*.

5.2. Finding an Intentional Explanation

The second place that PAGAN may find an explanation for an utterance is among the set of abstract planfors that it maintains. When the action to be explained is found as the first step of the plan of such a planfor (or as one of its possible first steps, if it is a partially ordered plan with more than one possible first step), then the goal of that planfor constitutes a possible explanation of the action. When I refer to an utterance 'matching' a planfor, I mean that it matches a plan step in this way. The process of finding planfor explanations for an utterance in long-term memory is called *planfor retrieval.*

Expectation matching is always preferable to finding an explanation that is unrelated to the previous dialogue. This is because people try to be coherent in their dialogues, and expectation matching reflects such coherence. Thus, like PAM (see Chapter 2), PAGAN always checks first for an explanation of an utterance in its dialogue model. Doing so causes PAGAN to prefer explanations that are grounded in what is known about the dialogue. PAGAN will look for an explanation among the abstract planfors it knows only if it does not find an expected action that matches the utterance. For example, consider the process of trying to explain the statement:

User: My file takes up 60 blocks.

If UC had just asked the question:

UC: How big is your file?

then it would make little sense to interpret the user's statement as a request to be told how to reduce the size of the file. The expectation of an answer to the question provides a more plausible explanation of the utterance.

5.3. Finding a Non-Intentional Explanation

The third place that PAGAN may find an explanation for an utterance is among the non-intentional explanations in its knowledge base [c.f. Wilensky 1983]. A *non-intentional explanation* is one that is not based on plans and goals. If PAGAN were processing the question:

User: How can I print a double-spaced draft of my paper?

it might reason as follows: The user wants to know how to print the draft because she wants to print it. She wants to print it because she wants to edit it off-line. At this point, the non-intentional explanation 'people often edit their papers' could be used to explain the goal of editing. Thus, the intentional reasons that the user might want to edit the file would not need to be inferred.

Themes, as described by Schank and Abelson [1977], are a kind of non-intentional explanation. Themes are concepts that "contain the background information upon which we base our predictions that an individual will have a certain goal" [p. 132]. For example, the 'preservation of resources' theme might provide a non-intentional explanation of the goal of preserving a particular file.

Non-intentional explanations are sought when expectation matching yields no explanations and PAGAN determines that it should no longer seek a planfor explanation of the utterance (see Chapter 8), or when no planfor explanation is extant. There are two reasons for treating non-intentional explanations as explanations of last resort. First, a non-intentional explanation generally does not meet the criterion of applicability as well as does an intentional explanation for the same event. The obvious exceptions to this are when the explanation chain being analyzed has already been determined to be inapplicable to the task of the parent system, or when no intentional explanation of the event can be found. The second reason for preferring intentional explanations is that they are more likely to meet the grounding criterion; the adoption of a non-intentional explanation assures the termination of the explanation chain, eliminating the opportunity for further grounding.

5.4. Implementation

Plan recognition systems have traditionally relied on structure matching to locate potential explanations of an event. Unfortunately, structure matching suffers from a variety of problems that make it inefficient for this task. In the following sections, I first describe the structure matching paradigm, and the problems with it. Next, I detail an alternative to matching that solves many of these problems. Finally, I show how this approach can be used to find the three classes of explanations for an utterance described above.

5.4.1. The Structure Matching Paradigm

The traditional approach to abductive inference for language comprehension encodes abductive inference rules as templates or schemas. In trying to locate rules that might be useful in explaining an input, such systems compare the input against a set of candidate templates. The comparison is performed by a matching algorithm such as unification. In this structure matching paradigm, the two structures are compared piece-by-piece to determine whether any incompatibilities exist between them. Most matchers allow the structures they are matching against one another to contain variables. These variables serve two purposes. First, they allow a concept such as a plan schema to be expressed in general terms that may account for a wide range of observed phenomena. Secondly, they are useful in requiring that two different portions of a pattern match exactly the same concept.

The structure matching paradigm is deficient as a tool for retrieving explanations in four ways:

1. The matcher must keep track of variable bindings
2. The boundaries of the pattern must be known in advance
3. The entire computational expense is incurred when matching commences
4. Partial results of failed matches are not cached

First, a matcher requires a complicated mechanism for keeping track of variable bindings as matching proceeds. In addition to being cumbersome, these bindings may have to be retracted if backtracking is used. Secondly, structure matching requires that the limits of the pattern to be matched (that is, the set of concepts that are part of the pattern) be determined before matching commences. Exactly and only those concepts that are deemed to be a part of the pattern may be compared during matching. Thirdly, the matching paradigm incurs its entire computational expense at the time of its explicit invocation. There is no provision for doing part of the work before a comparison is requested. Finally, all results of a failed match are lost, even when a portion of those results may be applicable to other patterns.

In the following sections, I detail a method that can be used to locate potential explanations for an utterance without the use of an explicitly-invoked pattern matcher. This technique works by making many individual categorization inferences based on highly localized knowledge. Not only does this method obviate the need for structure matching and variables, but it allows the categorization inferences to be made as knowledge is added to the system, instead of when an explanation is required.

5.4.2. The Local Categorization Paradigm

Matching in KODIAK is performed by two automatic processes that obviate the need for a goal analyzer to contain an explicit structure matcher. Together, these two processes facilitate explanation retrieval, without the deficiencies of the matching paradigm identified above. These processes operate on each concept at the time it is introduced into the knowledge base, not just to concepts being explained by a goal analyzer. The first process, *concretion*, places each new concept at its correct level in the knowledge hierarchy. Concretion inferences are discussed by Wilensky [1983] and Norvig [1983, 1987]. Concretion inferences that are restricted to the logical consequences of the knowledge hierarchy, called *classification inferences*, are discussed by Schmolze and Lipkis [1983]. The second process, *equation*, ensures that, where required, the same concept fills two separate roles. The equation process automatically enforces the maintenance of relations such as KODIAK's *equate* relations [Wilensky 1987a], or KL-ONE's *role value maps* [Brachman and Schmolze 1985].

5.4.2.1. Concretion

The process of placing an observed action within the action hierarchy is not as simple as associating particular concepts with particular words. It often requires that a special kind of inference be made called a *concretion inference*. A concretion inference is one that moves a concept downward in the knowledge hierarchy, based on the relations in which the concept takes part. For example, if the observed action is a **DELETION**, and the thing being deleted is a **DIRECTORY**, then the observed action may be moved downward from **DELETION** in the action hierarchy and placed beneath the **DIRECTORY-DELETION** concept.

The advantage of having a **DIRECTORY-DELETION** concept, and making appropriate concretions to it, is that knowledge specific to the deletion of a directory (such as that it can be accomplished with the **rmdir** command in UNIX) can be attached directly to that concept. This obviates the need for a process that discriminates between various types of **DELETION** at knowledge retrieval time, while at the same time ensuring that only applicable knowledge is retrieved. A diagram of the knowledge needed to make this kind of inference is shown in Figure 5.1.

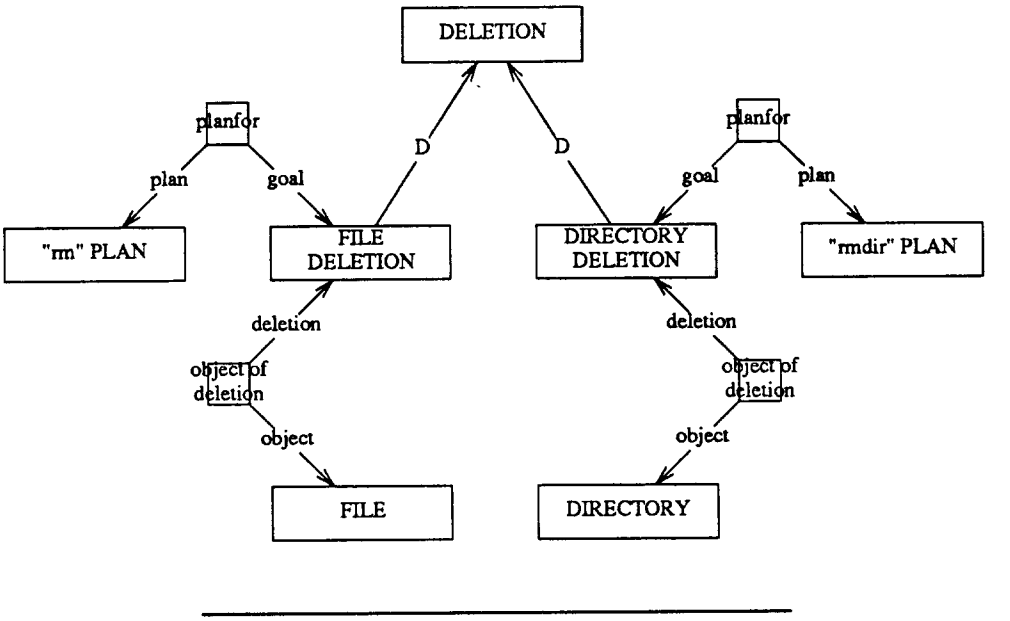

Figure 5.1. Knowledge Needed for Concreting **DELETION** to **DIRECTORY-DELETION**

5.4.2.2. Equation

The second automatic process is called *equation*. Equation is the process of inferring that a concept that plays one role also plays another. Consider the notion of a state change. A state change has two components, the before-state and the after-state. For example, a file deletion might be expressed as a state change from the file existing to the

file not existing.

It is unreasonable to compose a state change out of any two states. It would not for example make sense to say that there was a state change from George Bush being President to Jerome Garcia being Band leader. The reason is that to count as a state change, both the start state and the final state must be states of the same concept. The equation process is responsible for ensuring that consistency is maintained between objects that take part in certain relations. In this example, if the the George Bush concept were assigned to the object of the start state of a particular state change, the equation process would automatically assert that the object of the final state was also the George Bush concept.

Processing Example

The use of the automatic processes of concretion and equation can be seen in the following trace. It shows the parser's processing of the question:

User: How do I delete a file?

Each grammar rule is associated with a function that builds the semantics of the left hand side out of the semantics of the elements of the right hand side. Notice that as the semantics are being built, concretion and equation are automatically making inferences about them. For example, after the **TRANSITIVE-VP** is recognized, the **DELETION** concept is automatically concreted to be a **FILE-DELETION** because the object being deleted is a **FILE**.

User: How do I delete a file?

Applying semantics for **AUX => DO**
Applying semantics for **NP2 => I**
Applying semantics for **NP => NP2**
Applying semantics for **TRANSITIVE-VERB => DELETE**
Applying semantics for **ARTICLE => A**
Applying semantics for **NOMINAL => FILE**
Applying semantics for **NP2 => ARTICLE NOMINAL**
Applying semantics for **NP => NP2**
Applying semantics for **TRANSITIVE-VP => TRANSITIVE-VERB NP**

> *The parser is building the semantics for 'delete a file.' It asserts that the object being deleted is a file, and* **KODIAK** *then automatically concretes the* **DELETION** *to be a* **FILE-DELETION**. *It goes on to concrete the* **STATE-CHANGE-OBJECT** *slot of the deletion to be a* **STATE-CHANGE-OBJECT-OF-FILE-DELETION**.

Concreting **DELETION8** to be **FILE-DELETION**.
Concreting **STATE-CHANGE-OBJECT6** to be **STATE-CHANGE-OBJECT-OF-FILE-DELETION**.
Applying semantics for **VP2 => TRANSITIVE-VP**

Applying semantics for VP => VP2
Applying semantics for SENTENCE => NP VP

> *The* DELETE-ACTION *to which the user has referred is now concreted to be a* FILE-DELETE-ACTION.

Concreting DELETE-ACTION4 to be FILE-DELETE-ACTION.
Concreting CAUSES-OF-DELETE-ACTION6 to be CAUSES-OF-FILE-DELETE-ACTION.
Applying semantics for SENTENCE => HOW AUX SENTENCE

> *The* SPEAKER *and the* ACTOR *of a* TELL-ACTION *are represented as separate slots. Here they are automatically asserted to refer to the same concept.*

Equating the range of SPEAKER12 (which is ANIMATE)
with the range of ACTOR26 (which is USER).

> *Since the object of the* TELL-ACTION *is a question, the* TELL-ACTION *is concreted to be an* ASK-QUESTION. *Once this concretion is made, the concretion to* ASK-SPECIFICATION-QUESTION *can be made. This concretion cannot be done at the outset, because the automatic concretion mechanism moves downward through the hierarchy one link at a time.*

Concreting TELL-ACTION15 to be ASK-QUESTION.
Concreting UTTERANCE13 to be UTTERANCE-OF-ASK-QUESTION.
Concreting TELL-ACTION15 to be ASK-SPECIFICATION-QUESTION.
Concreting UTTERANCE13 to be UTTERANCE-OF-ASK-SPECIFICATION-QUESTION.

The parser produced the following interpretation:

```
Interpretation 1 (TELL-ACTION15):
(HAS-GOAL22 (DOMAIN USER)
 (RANGE
  (TELL-ACTION15 (DOMINATED BY ASK-SPECIFICATION-QUESTION) (ACTOR26 = USER)
   (HEARER13 = UNIX-CONSULTANT) (SPEAKER12 = USER)
   (UTTERANCE13 =
    (SPECIFICATION-QUESTION4 (DOMINATED BY SPECIFICATION-QUESTION)
     (WHAT-IS12 =
      (DELETE-ACTION4 (DOMINATED BY FILE-DELETE-ACTION) (ACTOR25 = USER)
       (CAUSES-OF-DELETE-ACTION6 =
          (DELETION8 (DOMINATED BY FILE-DELETION)
           (STATE-CHANGE-OBJECT6 = (FILE4 (DOMINATED BY FILE)))))))))))
```

5.4.2.3. Advantages of the Local Categorization Paradigm

The use of equation and concretion allows matching to be performed without the disadvantages inherent in the structure matching approach. First, there is no longer a need for variables. The first function of variables, to allow a wide variety of concepts to fill a particular slot, is now achieved by specifying the single concept in the knowledge

hierarchy that has the proper level of abstraction. For example, if the pattern must allow any human being to fill a particular slot, then the concept 'person' is used for that slot. The other function of variables, to ensure that two or more slots are filled by exactly the same concept, is now achieved by equating those slots, and allowing the equation process to enforce that rule.

The second criticism of the structure matching paradigm was that it requires that the boundaries of the pattern be known before matching commences. In contrast, the local categorization approach only needs to include in the definition of a concept C those related concepts R that directly contribute to C's definition; the definitions of the concepts in R are *not* part of the definition of C.

The third criticism of the matching paradigm was that it incurs its entire computational expense during explicit calls to the matcher. The local categorization paradigm performs its work in small pieces, as knowledge is added to the knowledge base. For example, when the knowledge is added that the object of a particular **DELETION** is a **FILE**, that **DELETION** can immediately be concreted to a **FILE-DELETION**.

Finally, the structure matching paradigm throws away partial results of failed matches. The processes of equation and concretion on the other hand register every inference they make permanently. These inferences are composed of single facts, each of which corresponds to a piece of a pattern that a structure matcher would use. Thus, the local categorization approach effectively caches portions of a match.

5.4.3. Finding a Known Explanation: Expectation Matching

Expectation matching is a subset of the process of grounding an utterance in what is already known of the speaker and of the dialogue. It aids in building explanations that are well-grounded by examining the utterance to determine whether it serves as a continuation of a plan that the speaker is known to be pursuing. If so, it is taken to be that expected action, and the goal that was already inferred to explain that action is an explanation of the utterance. In other words, an expectation is matched when the speaker is simply continuing some known plan. Expectation matching has been used in virtually every plan recognition system. For example, Wilensky's **PAM** story understanding system [1978] checked first in its representation of the story so far for an explanation of a new sentence. See also Allen and Perrault [1980], Pollack [1984], and Litman [1985] for various treatments of the problem of expectation matching. Note that I am not concerned here with the ways in which general inference processes (such as causal inference) mesh with expectation matching. See Bobrow et al. [1977] for an approach to this problem.

Expectation matching is tested by asserting that the action to be explained lies below (in the knowledge hierarchy) the next step of the most recently continued plan. An expectation is matched when the processes of concretion and equation find no conflicts based on this assertion. A conflict arises when two concepts that are in mutually exclusive portions of the knowledge hierarchy are equated, or are placed in a hierarchical relationship.

If an expected action is not matched in this way, that action's parent plan (that is, the plan in which the goal associated with the expected action plays a part) is itself examined as a possible source of an explanation. This process continues until an expected plan step is matched, or until all expected actions have been examined. PAGAN also checks in this manner for grounding by goal achievement. That is, after it checks a particular plan step for expectation matching, PAGAN performs the same check on its associated goal.

Because one dialogue participant is not obliged to continue a plan being pursued by another, PAGAN must monitor the progress of active plans. If an observed action is a continuation of an ongoing plan, nothing besides expectation matching need be done. If on the other hand a particular action is expected and a *different* action is observed, PAGAN must not only explain the new action, but it must also update its representation of previously inferred plans and goals to reflect this change in expectations. Of particular importance is the status of a plan initiated by UC in which the user is expected to, but has not yet, engaged. Because the user has not yet demonstrated a willingness to cooperate in such a situation, PAGAN must refrain from imputing the goal of such a plan to the user until the user is seen to continue the plan. Only then may PAGAN include the goal of the planfor as part of the user's goal structure. When the user is observed to continue such a plan, the user is said to have *adopted* that plan.

Processing Example

As an example of expectation matching, consider the following brief exchange:

> UC: Is the file in your directory?
> User: Yes.

The following trace shows PAGAN's processing of the user's utterance in the context of UC's question. PAGAN first tries to determine whether the user's response, TELL-ACTION17, is grounded in the preceding dialogue. The response does match the expectation of an answer to the question, and so that expectation is taken to be an explanation of the utterance.

UC: Is the file in your directory?
User: Yes.

The parser produced the following interpretation:

```
Interpretation 1 (TELL-ACTION17):
(HAS-GOAL22 (DOMAIN USER)
 (RANGE
  (TELL-ACTION17 (DOMINATED BY INDICATE-YES) (ACTOR21 = USER)
   (HEARER14 = UNIX-CONSULTANT) (SPEAKER13 = USER)
   (UTTERANCE14 = AFFIRMATIVE))))
```

UC's question has set up the expectation that the user will respond with yes or no. This expectation is represented as INDICATE-YES-OR-NO *below. The user's response of 'yes' is found to meet this expectation.*

Attempting to find an explanation for TELL-ACTION17.
  Determining whether TELL-ACTION17 is grounded in the preceding dialogue.
  Trying to ground TELL-ACTION17 in INDICATE-YES-OR-NO.
  TELL-ACTION17 was grounded.
Explanation found for TELL-ACTION17.

5.4.4. Finding a New Explanation: Planfor Retrieval

PAGAN uses planfors as abductive rules for inferring new explanations of an utterance. Doing so is a two-step process:

1. Selection of a planfor
2. Incorporation of the selected planfor into the dialogue representation

First, PAGAN must select a candidate planfor from among those in its knowledge base. Secondly, the chosen explanation must be incorporated into the representation of the dialogue. These steps are discussed in the following sections.

5.4.4.1. Finding Planfors in Long-term Memory

The use of localized categorization makes the task of locating potential planfor explanations for an utterance a simple one. When semantic interpretation is performed on the sentence, a representation of the meaning of the sentence is constructed. This representation is automatically moved downward in the hierarchy by the concretion mechanism until it reaches the most specific concept that is part of the meaning of the sentence. This process is similar to the action of the KL-ONE classifier [Schmolze and Lipkis 1983] in the PSI-KLONE system [Brachman and Schmolze 1985]. Each planfor that the system knows about is attached to the concept that represents the first step of its plan. Therefore, to retrieve all the planfors that might explain an utterance, PAGAN need only start at the node that represents the meaning of the sentence, and collect any planfors that are attached to that concept or to any of its ancestors. There is no need to do any comparison or matching at this point.

Processing Example

The following trace shows PAGAN's initial processing of the question:

User: How do I delete a file?

Of note in this trace is the fact that no concretion or equation occurs during planfor retrieval; all concretion and equation needed for planfor retrieval has already occurred during semantic analysis, or is done immediately *after* the previous planfor retrieval. Thus, planfor retrieval is relatively effortless.

User: How do I delete a file?

> *A number of concretions and equations are made automatically during semantic interpretation. These facilitate the subsequent search for planfor explanations of the utterance. In particular,* TELL-ACTION23 *is concreted first to be an* ASK-QUESTION, *then an* ASK-SPECIFICATION-QUESTION. *This allows planfors attached to* ASK-QUESTION *or* ASK-SPECIFICATION-QUESTION *to be immediately accessed.*

Concreting DELETION16 to be FILE-DELETION.
Concreting STATE-CHANGE-OBJECT13 to be STATE-CHANGE-OBJECT-OF-FILE-DELETION.
Concreting DELETE-ACTION12 to be FILE-DELETE-ACTION.
Concreting CAUSES-OF-DELETE-ACTION14 to be CAUSES-OF-FILE-DELETE-ACTION.
Equating the range of SPEAKER19 (which is ANIMATE)
with the range of ACTOR29 (which is USER).
Concreting TELL-ACTION23 to be ASK-QUESTION.
Concreting UTTERANCE20 to be UTTERANCE-OF-ASK-QUESTION.
Concreting TELL-ACTION23 to be ASK-SPECIFICATION-QUESTION.
Concreting UTTERANCE20 to be UTTERANCE-OF-ASK-SPECIFICATION-QUESTION.

The parser produced the following interpretation:

```
Interpretation 1 (TELL-ACTION23):
(HAS-GOAL38 (DOMAIN USER)
 (RANGE
  (TELL-ACTION23 (DOMINATED BY ASK-SPECIFICATION-QUESTION)
   (ACTOR29 = USER) (HEARER20 = UNIX-CONSULTANT) (SPEAKER19 = USER)
   (UTTERANCE20 =
    (SPECIFICATION-QUESTION5 (DOMINATED BY SPECIFICATION-QUESTION)
     (WHAT-IS17 =
      (DELETE-ACTION12 (DOMINATED BY FILE-DELETE-ACTION)
       (ACTOR28 = USER)
       (CAUSES-OF-DELETE-ACTION14 =
          (DELETION16 (DOMINATED BY FILE-DELETION)
           (STATE-CHANGE-OBJECT13 = (FILE14 (DOMINATED BY FILE)))))))))))
```

Attempting to find an explanation for TELL-ACTION23.
Determining whether TELL-ACTION23 is grounded in the preceding dialogue.
TELL-ACTION23 was not grounded in the preceding dialogue.

Here, a planfor explanation of TELL-ACTION23 *is sought.* PLANFOR4 *is found immediately, without any further categorization. A few equations are subsequently made, while building the new plan structure.*

Trying to find planfor explanations for TELL-ACTION23.
Found PLANFOR4 as an explanation for TELL-ACTION23.
Synopsis:
Goal: know something;
Step 1: ask;
Step 2: be told.
Equating the range of KNOWN12 (which is THING)
with the range of WHAT-IS17 (which is DELETE-ACTION12).
Equating the range of KNOWER12 (which is ANIMATE)
with the range of SPEAKER19 (which is USER).
Inferred goal is KNOWING-THAT6.

Explanation found for TELL-ACTION23.

Attempting to find an explanation for KNOWING-THAT6.
Determining whether KNOWING-THAT6 is grounded in the preceding dialogue.
KNOWING-THAT6 was not grounded in the preceding dialogue.

Once again, the correct planfor is retrieved without any matching at the time of retrieval. The equations are made only after the planfor has been found and instantiated.

Trying to find planfor explanations for KNOWING-THAT6.
Found PLANFOR8 as an explanation for KNOWING-THAT6.
Synopsis:
Goal: Achieve X;
Step 1: Know how to X;
Step 2: Do X
Equating the range of PLANFOR42 (which is EVENT)
with the range of CAUSES-OF-DELETE-ACTION14 (which is DELETION16).
Equating the range of CAUSES-OF-DELETE-ACTION14 (which is DELETION16)
with the range of PLANFOR42 (which is EVENT11).
Equating the range of PLANFOR42 (which is EVENT11)
with the range of CAUSES-OF-DELETE-ACTION15 (which is EVENT12).
Equating the range of CAUSES-OF-DELETE-ACTION15 (which is EVENT12)
with the range of PLANFOR42 (which is DELETION17).
Equating the range of PLANFOR42 (which is DELETION17)
with the range of CAUSES-OF-DELETE-ACTION15 (which is DELETION18).
Inferred goal is DELETION18.

Explanation found for KNOWING-THAT6.

5.4.4.2. Incorporating an Explanation into the Dialogue Representation

Once an explanation has been found, it must be attached to the representation of the dialogue, so as to ensure that the explanation is properly grounded. The ways that an explanation can attach to the model of the dialogue were discussed in Chapter 3. In particular, an explanation may be integrated into the dialogue representation so as to continue a plan, to delay a plan, to reject a plan, to achieve a goal, to question a goal, or to reject a goal. Continuation of a plan and achievement of a goal have already been discussed in the section above on expectation matching. The other types of integration are performed by PAGAN only when a higher-level plan is continued (as illustrated in the processing example below). See Litman and Allen [1984] for a description of how these types of integration can be implemented in the general case.

Processing Example

This trace shows PAGAN's processing of the user's statement in the following exchange:

> UC: Is the file in your directory?
> User: The file is in /tmp.

There are several points to note in this trace. PAGAN's first step is to try to match the semantic interpretation of the utterance (TELL-ACTION21) against the expectation of a response to UC's question. When this fails, PAGAN checks for plan rejection. Since the utterance does not directly match the goal initiated by UC in its question, a planfor explanation is sought for the utterance. PAGAN finds such an explanation, and applies the algorithm recursively. This time, it finds that the inferred goal (KNOWING-WHETHER4) matches the goal that UC had initiated. Thus, the user has provided a different method for achieving UC's goal. UC's original plan is therefore rejected. Notice that the matching is done simply by attempting to concrete the action to be explained to be an instance of the concept to be matched. If the concretion is successful, then the match is successful; if the concretion fails, then the match is unsuccessful.

UC: Is the file in your directory?

User: The file is in /tmp.

Interpretation produced by the parser: TELL-ACTION21

Attempting to find an explanation for TELL-ACTION21.
Determining whether TELL-ACTION21 is grounded in the preceding dialogue.

Here, PAGAN *tries to determine whether* TELL-ACTION21 *is a continuation of* UC*'s plan, which calls for a yes/no answer at this point. It does so by asserting that it is, then catching the error that results.*

Trying to ground TELL-ACTION21 in INDICATE-YES-OR-NO.
Concreting TELL-ACTION21 to be INDICATE-YES-OR-NO.
TELL-ACTION21 was not grounded in INDICATE-YES-OR-NO because:
Can't concrete UTTERANCE18 to UTTERANCE-OF-INDICATE-YES-OR-NO because the ranges differ.

Next, PAGAN *checks whether* TELL-ACTION21 *constitutes a new plan for* UC*'s existing goal of knowing whether the file is in the user's directory. It does this in the same way that it checked for plan continuation above.*

Determining whether TELL-ACTION21 is a new plan for the existing goal of KNOWING-WHETHER4
TELL-ACTION21 is not a new plan for the existing goal of KNOWING-WHETHER4 because:
Attempt to concrete incompatibly from TELL-ACTION21 to KNOWING-WHETHER4.
TELL-ACTION21 was not grounded in the preceding dialogue.

Now, PAGAN *looks for a planfor explanation of the utterance. It finds one, and infers that the user has the goal of* UC *knowing whether the file is in the directory.*

Trying to find planfor explanations for TELL-ACTION21.
Found PLANFOR3 as an explanation for TELL-ACTION21.
Synopsis:
Goal: hearer know whether X;
Step 1: Tell hearer that X.
Inferred goal is KNOWING-WHETHER5.

Explanation found for TELL-ACTION21.

Attempting to find an explanation for KNOWING-WHETHER5.
Determining whether KNOWING-WHETHER5 is grounded in the preceding dialogue.

Once again, PAGAN *checks for plan continuation.* KNOWING-WHETHER5 *is not a type of indicating yes or no, so the check fails.*

Trying to ground KNOWING-WHETHER5 in INDICATE-YES-OR-NO.
KNOWING-WHETHER5 was not grounded in INDICATE-YES-OR-NO because:
Attempt to concrete incompatibly from KNOWING-WHETHER5 to INDICATE-YES-OR-NO.

Now, PAGAN *checks whether* KNOWING-WHETHER5 *represents a new plan for* UC's *existing goal. It does, so the old plan is rejected, and processing terminates.*

Determining whether KNOWING-WHETHER5 is a new plan for the existing goal of KNOWING-WHETHER4
KNOWING-WHETHER5 is a new plan for the existing goal of KNOWING-WHETHER4
Existing plan PLAN49 has been rejected; new plan is PLAN53.
Explanation found for KNOWING-WHETHER5.

5.4.5. Finding Thematic and Non-Intentional Explanations

The third type of explanation available to PAGAN is a thematic or non-intentional explanation. Like planfor explanations, thematic explanations can be associated with particular concepts in the knowledge hierarchy. When an action is to be explained, thematic and non-intentional explanations can be sought at the same time that planfor explanations are being sought. Thus, localized categorization also allows the retrieval of themes without the use of structure matching.

If only a thematic or non-intentional explanation is found, it is used as the explanation of the utterance, and no more processing need be done. If both a planfor explanation and a non-intentional explanation is found, the rules for deciding whether to continue the analysis (described in Chapter 8) are used to select one of the explanations. If analysis is to continue, the planfor explanation is selected; if analysis is to halt, the non-intentional explanation is chosen.

Processing Example

Here is a simple example of an utterance that matches a common non-intentional explanation: that people like to talk about the weather:

User: The weather is nice.

PAGAN doesn't check for a planfor explanation of this utterance, because it determines that the explanation is complete (see Chapter 8 for an explanation of how PAGAN decides whether an explanation is complete). PAGAN then looks for a theme to explain the utterance, and finds the 'talk about the weather' theme. It uses this theme as an explanation of the utterance.

User: The weather is nice.

Interpretation produced by the parser: TELL-ACTION15

Determining whether to chain on TELL-ACTION15.
  Determining whether TELL-ACTION15 is a complete explanation of (THE WEATHER IS NICE).
    Determining whether TELL-ACTION15 is outside of UC's domain of expertise.
    TELL-ACTION15 is outside of UC's domain of expertise.
    Determining whether TELL-ACTION15 is explained by a theme.
    TELL-ACTION15 is explained by the TALK-ABOUT-WEATHER-THEME.

After determining that talking about the weather is outside of UC*'s domain of expertise, the utterance is immediately connected to the theme of talking about the weather.*

  The explanation is complete.
Chaining should not be done on TELL-ACTION15.

5.5. Summary

An explanation of an utterance (or of an inferred goal) can be found either among the expected actions found in the dialogue model, among the set of intentional explanations in the knowledge base, or among the set of non-intentional explanations in the knowledge base. The retrieval of each of these types of explanation is facilitated by a matching paradigm that focuses on making many local categorizations instead of matching large structures against one another. This local categorization paradigm is implemented by two automatic processes called *concretion* and *equation*. The use of these processes allows explanations to be retrieved directly, without large-scale structure matching.

Chapter 6

Sources of Ambiguity

It is possible for a speaker to say something that legitimately has more than one interpretation, and that a human hearer cannot disambiguate. Consider for example:

User: How can I save my program in a file?

Unless some discourse has preceded this utterance that gives some clue as to the referent of the word *program*, it is impossible to distinguish which of the following two interpretations the speaker intended:

1. How can I save the source code of my program in a file?
2. How can I save the executable code of my program in a file?

This is *real ambiguity*. Real ambiguity is a property of an utterance set in a particular context. Any goal analysis algorithm, when presented with an utterance that exhibits real ambiguity, should detect such an ambiguity.

Given that a speaker does not intend to be ambiguous, it is rare that real ambiguity exists in an utterance. This is because 'context' (a combination of knowledge of the speaker, of the dialogue, and of the world) typically combines with knowledge of the sentence uttered to promote one particular interpretation of the utterance. Under these circumstances, if a system attempting to understand an utterance notices an ambiguity, it is because that system failed to take into account some portion of the applicable context. This kind of ambiguity is called *resolvable ambiguity*. Resolvable ambiguity occurs when two or more interpretations of an utterance that does not exhibit real ambiguity exist concurrently at some point during the processing of that utterance. For example, the second utterance in the sequence:

User: I just typed my program into the vi editor.
User: How can I save my program in a file?

is open to the same two interpretations as is the previous example. However, the first utterance in the sequence provides a context against which this ambiguity may be resolved. Thus, if the statement appears ambiguous, it does so only briefly during the time that it is being processed; there is no ambiguity in its final interpretation.

It is important to note that resolvable ambiguity is a byproduct of an algorithm, not a property of an utterance. Given the same utterance to be explained, different algorithms may encounter different resolvable ambiguities. However, because regularities exist in the use of language, resolvable ambiguities that reflect those regularities will be noticed by any goal analysis algorithm that is engineered around them.

This chapter is concerned with the ways that resolvable ambiguity can arise during goal analysis. For the purposes of the rest of this dissertation, 'ambiguity' will refer to resolvable ambiguity unless explicitly stated otherwise. In this chapter, I describe several phenomena that may be encountered during goal analysis that can give rise to ambiguity, and show how ambiguity can be detected (or, where appropriate, how its detection can be avoided). In Chapter 7, I describe two methods for handling ambiguity: resolving it, and disregarding it.

There are three classes of ambiguity with which any goal analyzer must contend:

1. Sentence ambiguity
2. Utterance ambiguity
3. Goal ambiguity

Sentence ambiguity is ambiguity that arises when the sentence to be analyzed has more than one interpretation independent of its use. Utterance ambiguity is ambiguity that is inherent to the utterance being analyzed when set in a particular context. Goal ambiguity is ambiguity that arises when more than one known plan includes the action to be explained, and consequently there is more than one goal that might explain it. Much research has been devoted to detecting and resolving several subcategories of these types of ambiguity. Classes of ambiguity that have received much attention in AI, linguistics, philosophy, and psychology are lexical ambiguity, structural ambiguity, and referential ambiguity. My interest in these forms of ambiguity stems from the fact that, while many techniques are available for resolving such ambiguities (see Small et al. [1988] for a variety of approaches), not all ambiguities are resolvable prior to goal analysis. Thus the goal analyzer must be able to handle cases in which its input is ambiguous.

6.1. Sentence Ambiguity

Sentence ambiguity is ambiguity that is intrinsic to a sentence, independent of the context of that sentence. The two most common forms of sentence ambiguity:

1. Lexical ambiguity
2. Structural ambiguity

Words or phrases that have more than one sense can give rise to sentence ambiguity that is called *lexical ambiguity*. Under the rubric of lexically ambiguous terms, I include metonymic references, misspelled words with multiple plausible corrections, and, when the input is speech rather than written text, homonyms. The second major kind of sentence ambiguity is *structural ambiguity*. Structural ambiguity arises when there is more than one possible parse tree for a single sentence. Each of these types of ambiguity can be attributed to a sentence independent of its use in an utterance. Other types of sentence ambiguity have also been discussed in the literature, as for example ambiguity of scope. Birnbaum [1985] points out that to properly handle lexical ambiguity, memory and inference techniques must be integrated into language analysis. **PAGAN** represents a step in this direction.

6.2. Utterance Ambiguity

Utterance ambiguity is ambiguity that is inherent to the utterance being analyzed when set in a particular context. That is, a sentence with a single interpretation independent of context that may mean different things when used in different contexts exhibits utterance ambiguity. There are three kinds of utterance ambiguity:

1. Referential ambiguity
2. Categorization ambiguity
3. Topic ambiguity

Referential ambiguity arises when more than one potential referent for a concept is hypothesized at some point in the analysis of an utterance. Categorization ambiguity stems from the existence of more than one subcategory into which a concept of a particular category may be placed. Topic ambiguity arises when more than one concept might serve as the primary focus of an utterance. These types of ambiguity are discussed in the following sections.

6.2.1. Referential Ambiguity

The first class of utterance ambiguity, called *referential ambiguity*, involves problems of reference. For example, consider the utterances:

User: My directory has a file in it that I don't want.
User: How can I get rid of it?

In this example, the word 'it' in the second sentence could refer either to the file, or to the directory. There is no feature within the second sentence that can be used to isolate the correct referent. Although a focus mechanism [Grosz, 1978] will narrow the field to two possible referents, the file and the directory, it cannot by itself resolve the reference.[1]

[1] To see this, imagine that the second utterance had instead been "How can I clear it out?"

Thus, when goal analysis commences, the ambiguity persists.

6.2.2. Categorization Ambiguity

The second kind of utterance ambiguity, called *categorization ambiguity*, arises when an utterance that is not ambiguous according to the above classes of utterance ambiguity, nonetheless can be placed in more than one taxonomic category within the knowledge hierarchy. That is, an utterance exhibits categorization ambiguity when one of the components of the semantic interpretation of that utterance can be further specified in two or more competing ways. For example, consider the question:

User: What does ls do?

Given that the semantic interpreter can interpret 'ls' to mean 'use of the ls program,' there are two possible meanings of this utterance:

1. What does use of ls cause to happen?
2. What is the use of ls typically intended to cause?

This ambiguity occurs because of the use of the verb 'do.' It is possible to treat this ambiguity as lexical ambiguity of 'do.' However, each of these senses of 'do' has a common component; this common component is effectively what 'do' means (in Wilensky's terminology [1987b], this is the 'primal content' of the word). Thus, it is better to treat this example as categorization ambiguity in which the concept representing 'doing' can be concreted to one of two plausible subconcepts.

There are many concepts that are subject to categorization ambiguity. I will discuss one other example of categorization ambiguity, which is derivative of a taxonomy of question types. This taxonomy is useful in its own right, and it provides a good example of how categorization ambiguity arises. In the following sections, I first describe the taxonomy of question types, then show how it can give rise to categorization ambiguity.

6.2.2.1. A Taxonomy of Question Types

Previous question taxonomies [Lehnert 1977, Ram 1987] have focused on the content of a question as the basis for distinctions between question types. In contrast to this approach, the following taxonomy is based solely on the type of knowledge being requested. It is not a complete taxonomy; it does not handle questions that involve deictic reference. Nevertheless, it does provide a good example of categorization ambiguity.

There are three types of non-deictic questions a speaker may ask:

1. Specification questions
2. Description questions
3. Verification questions

A *specification question* is one in which the speaker requests that the hearer provide a *specification* of the concept identified in the question. A specification of a concept is a concept lying below that concept in the knowledge hierarchy.[2] For example, the question:

User: How do I delete a file?

is a specification question; its speaker is requesting that the hearer provide a specific method for deleting a file. A proper response might be:

UC: Use rm.

Here, the use of rm is a specific method for deleting a file (i.e. it lies below the concept 'method for deleting a file' in the knowledge hierarchy). Thus, a specification question is one where the questioner has restricted the answer to a particular category in the knowledge hierarchy.

As another example of a specification question, consider the exchanges:

Pat: What kind of doctor is she?
Jerry: A cardiologist.

Pat: What kind of doctor is she?
Jerry: A very good one.

This question is ambiguous, because there are many possible dimensions along which the type of doctor can be selected (in fact, the ambiguity is categorization ambiguity). However, no matter which interpretation is selected, the question is a specification question; it asks for a further specification along the chosen dimension.

A *description question* is one in which the speaker requests that the hearer describe a particular concept in terms of related concepts in the knowledge base. An example of a description question is:

User: What is a directory?

---

[2] References to 'the knowledge base' or 'the knowledge hierarchy' in this section mean 'the knowledge base of the hearer.' This includes facts that the hearer may infer from the knowledge base, whether or not they are explicitly represented there.

In this question the user is asking that the hearer describe knowledge relating to directories. An appropriate response might be:

UC: A directory is like a container for files.

Here the description of knowledge about directories is expressed analogically.

Another example of a description question is the following:

Beulah: What's two plus two?

This question is a description question because it is asking for a different description of a quantity that has already been identified in one way. However, the type of description typically used for a response to a question such as this one is highly conventional. The response 'an even number' is correct, but it is typically regarded as a poor description of the answer.

Note that another way to view description questions is as questions about *tokens* rather than about *concepts*. However, handling description questions in this manner would require that the semantic analyzer constantly consider whether each piece of the utterance should be parsed, or should be treated as a token. Thus, it is more efficient to treat description questions as questions about *concepts*, where the poser of the question is unsure about how the concept relates to other concepts.

A *verification question* is one in which the speaker requests that the hearer indicate whether a particular concept is a part of the knowledge base. For example:

User: Do I need write permission on a file to delete it?

is a verification question. An appropriate response would be:

UC: No.

Alternative questions are effectively compound verification questions, although they may presuppose a particular background. For example, the question:

User: Is a process a file or a directory?

is a combination of the questions 'is a process a file?' and 'is a process a directory?' with the presupposition that a process is one of the two.

6.2.2.2. Question Miscategorization

Ambiguity can arise whenever it is possible to miscategorize a question within this taxonomy. There are two types of miscategorizations of questions that are easily made. Consider the question:

User: What is a way to delete a file?

When interpreted the natural way, as a specification question, an appropriate response might be:

UC: Use rm.

The response:

UC: It's a method by which one can make an existing file non-existent.

would be inappropriate. The hearer who made this response would have misinterpreted the question as a description question.

The opposite type of misinterpretation, taking a description question to be a specification question, can also occur. Consider the question:

User: What is a directory?

The normal interpretation of this question is that it is a description question. A normal answer to the question might be:

UC: A directory is a place where many files may be kept.

However, if the question is interpreted as a specification question, a response might be:

UC: /usr is a directory.

Each of these types of misinterpretation is an example of categorization ambiguity. The parent category is the 'question' concept. The ambiguity arises in these examples because this concept might reasonably be concreted to either the 'specification question' concept, or to the 'description question' concept.

Although specification questions and description questions in general require different kinds of answers, it is possible to use specification to respond to a description question. For example, the question:

Novice Cardplayer: What is a suit?

in the context of card playing might appropriately draw the response:

Sharif: It's one of clubs, diamonds, hearts, or spades.

This type of 'description by example' is often used when one finds it difficult to provide any other kind of description. Since other types of description are usually more powerful, they are typically preferred.

The distinction between specification questions and description questions is different from Fauconnier's distinction between properties of roles and properties of values of roles [1985]. Fauconnier points out that certain noun phrases can refer either roles or to individuals. Consider the question:

Who is the President of the United States?

In the role interpretation, 'the President' means the office of the President; appropriate responses to the question under this interpretation might be "the head of the executive branch of the government" or "the leader of the free world." In the role value interpretation, 'the President' means the holder of the office; appropriate responses to the question under this interpretation might include "George Herbert Walker Bush" or "Barb's hubby" or "That guy over there."

This question demonstrates that the specification/description distinction is orthogonal to Fauconnier's role/role value distinction. Regardless of whether 'the President' is given a role interpretation or a role value interpretation, the question is a description question. That is, the question demands an answer that *describes* the term 'the President' under either interpretation. The possible answers listed above are all descriptions. For a response to be a specification, it would have to lie below the concept 'the President' in the knowledge hierarchy. Each of the responses is somehow related to this concept, but not hierarchically.

6.2.3. Topic Ambiguity

The final type of utterance ambiguity is called *topic ambiguity*. Topic ambiguity arises when there are several possible concepts within the representation of the utterance whose *level of specificity* might be in question. The level of specificity of a concept is the depth of that concept within the knowledge hierarchy. For example, the concept **CLOTHING** is not very specific (it has many descendants), while the concept **PLAID-POLYESTER-PANTS** is much more specific.

When a hearer builds a representation of the meaning of a speaker's utterance, each concept within that representation may differ in its level of specificity from the concept the speaker used in creating the utterance, or from the level of specificity that the speaker wants to know about the concept. A speaker often intends that the hearer recognize that a particular concept is overspecified or underspecified in this way. I call this concept the *topic* of the utterance. For example, the topic of the question:

User: How do I delete a file?

is in most contexts the concept 'method for file deletion,' because that concept has been *underspecified* relative to the concept the user wants the consultant to express. In asking this question, the user has expressed a concept that is underspecified relative to the desired concept, and is asking for a more specific concept lying below 'method for file deletion' in the knowledge hierarchy. One such concept is the 'rm-method' concept.

I use the term *topic* to indicate that concept whose level of specificity is of primary importance to the meaning of the utterance. As such, my use of the term topic is typically the same as the notion of topic found in the *topic/comment* (or alternatively, *given/new*) distinction (See Levinson [1983] for a discussion of these distinctions). The *given* of a sentence typically refers to the concept that is currently the *discourse topic*. It is usually this topic whose level of specificity is in question. Consider for example the following sentence, in which italics is used to indicate emphasis and therefore the *given* of the sentence:

User: Did *Karen* delete the file?

Karen is also the *topic* (in my sense) of the sentence, since it is only the specification of Karen that is in question.

Topic ambiguity occurs when the hearer has difficulty determining which concepts the speaker intends to be recognized as being overspecified or underspecified. As an example of topic ambiguity, consider a question such as:

User: Does mv copy files?

Carberry [1983] points out that questions such as this one are ambiguous. This question could either be asked as a way to find out what **mv** does, or as a way to find out how to copy files (I am ignoring for the moment the categorization ambiguity between doing as *causing*, and doing as *typical function*). When used in the first way, the utterance indicates a suspicion on the part of the speaker that **mv** might copy files. When used in the second way, the utterance indicates a suspicion on the part of the speaker that a way to copy files might be **mv**. Thus, in a verification question such as this one, one of the concepts within the representation of the question has been overspecified relative to what the speaker knows to be true. The ambiguity arises because there is more than one concept that might play the role of this overspecified question topic.

Topic ambiguity is not restricted to verification questions. Rather, it can arise whenever the relationship between two concepts is discussed. However, while topic ambiguity can occur in any utterance, its severity differs from utterance to utterance. That is, the consequences of selecting the wrong interpretation are graver in some situations than in others. It is most important to correctly analyze topic in the following three categories of utterance:

1. Verification questions
2. Specification questions
3. Negations

While topic ambiguity does not derive from these classes of utterance, it is important to detect topic ambiguity when an utterance falls into one of these classes.

The occurrence of topic ambiguity in verification questions was discussed above. The second type of utterance in which it is important to check for topic ambiguity is a specification question. In a specification question, one of the concepts expressed by the speaker is underspecified relative to what the speaker wants to know about the concept. Ambiguity can arise when it is unclear which concept is underspecified in this way. For example, in a question such as the following one, the concept that initially appears to be the topic of the question is not the intended topic:

User: How can I delete 300 blocks?

The user with a disk quota problem might ask this question in an attempt to determine not how to delete files, but rather which files should be deleted. The problematic part of the processing of this question is to avoid giving an answer like:

UC: Use rm.

This answer would only be appropriate if the speaker did not know how to delete files in general. However, in this context, it is likely that the speaker does know how to delete files, and is interested in finding out which files she may delete so as to have freed 300 blocks. An appropriate response might then be:

UC: Do you have any core files?

or:

UC: Delete the file gihugic.bak in your home directory.

The claim here is that in this context, the question is *not* a question about deleting; rather, it is about a file or files of 300 blocks that might be deleted. Because the concept that is to be specified is not necessarily the one that the initial interpretation of the question would indicate, specification questions may exhibit topic ambiguity.

Finally, negations provide a significant source of topic ambiguity. When a speaker uses a negation, it means that one of the expressed concepts is incorrectly specified relative to what the speaker believes to be true. Determining which concept is the topic of that negation though can be problematic. Consider the statement:

User: I don't want to delete my file with uncompress.

The user might have said this either because she was concerned that use of uncompress would delete her file, or because she wanted to delete her file with something other than uncompress. In the first interpretation, the desired result of using uncompress is incorrectly specified; in the second interpretation, the plan for deleting the file is incorrectly specified. Note that there are more interpretations for this utterance (*I* don't want to do it, *Barney* does; I want to delete a *directory* with uncompress; etc.). However, these two interpretations are adequate for the purpose of examining topic ambiguity.

In summary, verification questions, specification questions, and negations all focus on a single topic whose level of specificity is in question. Verification questions ask whether their topic has been correctly overspecified. Specification questions ask for further specification of their topic. Negations assert that their topic has been incorrectly specified. Topic ambiguity arises when there is more than one concept that might serve as the topic in question.

6.3. Goal Ambiguity

Goal ambiguity is ambiguity that occurs when a single action can be a plan for more than one goal. In processing terms, this means that the utterance or goal to be explained matches more than one planfor in long-term memory. For example, the question:

User: Do you know how to print a file on the imagen?

can be explained by two planfors, one describing the direct speech act, the other describing the indirect speech act.

It is instructive to analyze goal ambiguity along two separate axes:

1. According to the types of goals inferred
2. According to the relationship between the alternative explanations

First, the inferred goals can be either discourse goals or domain goals. At the discourse level, one of the most widely-explored forms of goal ambiguity involves direct and indirect speech acts [Austin 1962, Searle 1969, Allen 1979]. The utterance:

User: I'd like you to tell me how to edit a file.

for example could either be used to indicate a desire (the direct interpretation), or to request an action (the indirect interpretation). I term this kind of goal ambiguity *direct/indirect ambiguity*.

Goal ambiguity can also be seen at the level of the domain of discourse. For example, if the user has the goal of deleting a file, that goal may be explained by the goal of cleaning up a directory, or of freeing up disk space, or of making the data in the file inaccessible, or of freeing up the filename, or some combination of these goals. Each of these explanations can be represented by a distinct planfor, and it is the existence of more than one such planfor that causes the goal to exhibit goal ambiguity.

The point of making a distinction between dialogue-level and domain-level goal ambiguity is not to claim that these two kinds of ambiguity require different approaches for their resolution (as suggested by Allen, Frisch, and Litman [1982] for example). To the contrary, the point is that even though there is a strong tendency to separate dialogue phenomena from domain phenomena, nevertheless dialogue-level goal ambiguity and domain-level goal ambiguity should be handled in the same way for the purposes of disambiguation. In other words, it is unimportant to distinguish between discourse goals and domain goals; both types of goals can be handled by a single mechanism.

Goal ambiguity can also be codified into *vertical ambiguity* and *horizontal ambiguity*, according to the relationship between the alternative explanations. Consider the request:

> **User:** Can you tell me how to change my password?

This utterance matches the planfor that represents the fact that asking whether someone can perform a task is a plan for getting her to do the task. It also matches the more general planfor that states that a way to request that someone perform a task is to inquire as to whether a precondition of that task holds. Because the former planfor is derivative of the latter one, and is therefore a descendant of it in the knowledge hierarchy, I use the term *vertical ambiguity* to describe this sort of goal ambiguity.

Horizontal ambiguity is goal ambiguity in which neither of the competing planfors lies above the other in the knowledge hierarchy. Consider the planfor that indicates that a way to find out whether a fact holds is to ask whether it holds. This planfor can also be used to explain the above question; it corresponds to the direct interpretation of the question. Since it lies neither above nor below the planfors identified in the above discussion on vertical ambiguity, it is horizontally ambiguous with each of them. These examples of vertical and horizontal goal ambiguity are shown in Figure 6.1.

It is possible to avoid vertical ambiguity completely through suitable construction of the planfor knowledge base in long-term memory. However, it would hardly be desirable to handle the problem of vertical ambiguity in this manner. This is because general planning knowledge and specific, frequently-used planning knowledge are both important for efficient processing. Specific knowledge allows frequently used plans to be handled quickly, and allows additional information that is not applicable to the more general plan to be attached directly to the plan to which it is applicable. General planning knowledge on the other hand provides a way to handle a wide range of less commonly used plans in a minimum of space. Thus, it is desirable to maintain both kinds of knowledge in the knowledge base.

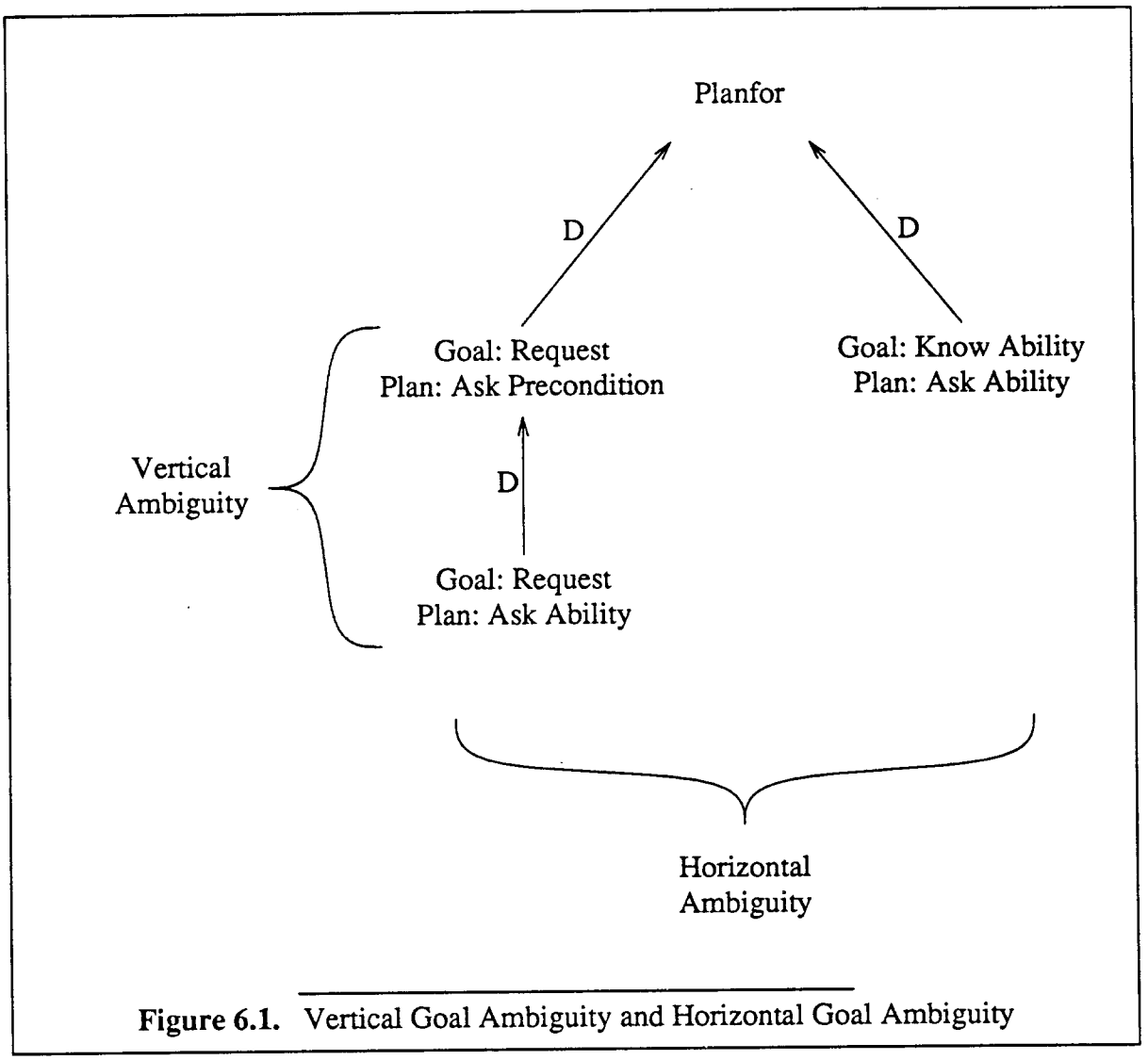

Figure 6.1. Vertical Goal Ambiguity and Horizontal Goal Ambiguity

6.4. Implementation -- The Detection of Ambiguity

The detection of ambiguity is a byproduct of the task of creating a pool of interpretations of an utterance large enough to ensure that it contains the correct interpretation. Detecting ambiguity is therefore not a desirable thing; if a program could always retrieve only the single candidate that was the correct interpretation, it would never detect ambiguity, and yet would always correctly explain its input. A program that considered all available sources of knowledge *simultaneously* would avoid the detection of ambiguity in this way. However, in practice, it is not possible to consider all sources of knowledge at once. Consequently, a goal analysis program must consider one source of knowledge at a time, notice resolvable ambiguities (so as to ensure that the correct interpretation is among those under consideration), and resolve the ambiguities based on further sources of knowledge. The following sections describe how the search for plausible explanations gives rise to each of the types of ambiguity discussed above, and how PAGAN can determine whether the explanation chains under consideration are compatible or incompatible

with one another.

6.4.1. Detecting Lexical, Structural, and Referential Ambiguity

Techniques for the detection of lexical, structural, and referential ambiguities are well-known and straightforward. The knowledge needed for the detection of lexical ambiguity can be encoded as multiple entries for the word in the lexicon, as multiple grammar rules for the word (when a lexicon is not used), or as multiple meaning associations in the semantic interpreter's knowledge base. The knowledge needed for the detection of structural ambiguity is encoded as grammar rules. The knowledge required for the detection of referential ambiguity can be maintained in the semantic interpreter's knowledge base, or in a separate knowledge base such as a focus mechanism [Grosz 1978]. Detection of these three kinds of ambiguity is then simply a matter of allowing the parsing and semantic interpretation algorithms to produce as many outputs as they find interpretations for the utterance.

6.4.2. Detecting Categorization Ambiguity

Categorization ambiguity can be detected in two ways. Since categorization ambiguity is caused by the existence of multiple possible competing concretions of a particular category, the natural time to detect it is during semantic interpretation. Consider again this example of question-type ambiguity:

User: What is a way to delete a file?

Categorization ambiguity arises in this example because the concept that represents the utterance, QUESTION, might reasonably be concreted either to SPECIFICATION-QUESTION or to DESCRIPTION-QUESTION. Thus, the ambiguity can be introduced by creating two explanations of the utterance, one of type SPECIFICATION-QUESTION, the other of type DESCRIPTION-QUESTION. Note that the parent category need not exist prior to the processing of the utterance, as long as the knowledge representation language can properly incorporate the new concept into the knowledge hierarchy (see Travis and Jackson [1988] for a logic-based approach to categorization that will solve this problem under a closed-world assumption).

In practice though, it is often simpler to introduce categorization ambiguity during parsing. In this approach, two separate grammar rules are included for the same construct. These rules have separate semantic interpretation procedures associated with them; one procedure produces an interpretation based on the first competing category, the other produces an interpretation based on the second competing category. A variant on this approach is to have only one grammar rule whose semantic interpretation procedure produces both interpretations. The difference between these two methods for treating categorization ambiguity corresponds roughly to the distinction between a word with multiple word senses and a word that is vague (See Lytinen [1988] for a discussion of

vague words). Often, the easiest way to handle vague words in practice is to provide them with distinct word senses.

Processing Example

The following trace shows the initial stages of the processing of the utterance:

User: What is a way to delete a file?

The parser detects categorization ambiguity in this example via multiple grammar rules for the 'what is' construct. It builds two separate interpretations of the utterance, one based on the concept **SPECIFICATION-QUESTION**, the other on the concept **DESCRIPTION-QUESTION**. These two interpretations are then passed to **PAGAN**, which must resolve the ambiguity.

User: What is a way to delete a file?

The first interpretation of this question is that it is a normal wh-question that should be treated as a specification question.

Applying semantics for **WH => WHAT**
Applying semantics for **TO-BE => IS**
Applying semantics for **TRANSITIVE-VERB => DELETE**
Applying semantics for **ARTICLE => A**
Applying semantics for **NOMINAL => FILE**
Applying semantics for **NP2 => ARTICLE NOMINAL**
Applying semantics for **NP => NP2**
Applying semantics for **TRANSITIVE-VP => TRANSITIVE-VERB NP**
Applying semantics for **VP2 => TRANSITIVE-VP**
Applying semantics for **VP => VP2**
Applying semantics for **NP => A WAY TO VP**
Applying semantics for **SENTENCE => WH TO-BE NP**

The second interpretation of the question uses the 'what is' construct. The question is taken to be a description question.

Applying semantics for **TO-BE => IS**
Applying semantics for **TRANSITIVE-VERB => DELETE**
Applying semantics for **ARTICLE => A**
Applying semantics for **NOMINAL => FILE**
Applying semantics for **NP2 => ARTICLE NOMINAL**
Applying semantics for **NP => NP2**
Applying semantics for **TRANSITIVE-VP => TRANSITIVE-VERB NP**
Applying semantics for **VP2 => TRANSITIVE-VP**
Applying semantics for **VP => VP2**
Applying semantics for **NP => A WAY TO VP**
Applying semantics for **SENTENCE => WHAT TO-BE NP**

Notice that the TELL-ACTION *in the first interpretation is a specification question, while in the second interpretation it is a description question. In all other respects, the two representations have the same structure.*

The parser produced the following interpretations:

Interpretation 1 (TELL-ACTION22):

```
(HAS-GOAL38 (DOMAIN USER)
 (RANGE
  (TELL-ACTION22 (DOMINATED BY ASK-SPECIFICATION-QUESTION) (ACTOR34 = USER)
   (HEARER20 = UNIX-CONSULTANT) (SPEAKER19 = USER)
   (UTTERANCE20 =
    (SPECIFICATION-QUESTION6 (DOMINATED BY SPECIFICATION-QUESTION)
     (WHAT-IS16 =
      (THING1 (DOMINATED BY THING)
       (PLANFOR49* =
         (PLAN74 (DOMINATED BY PLAN)
          (PLAN-STEP67 =
           (DELETE-ACTION6 (DOMINATED BY DELETE-ACTION)
            (CAUSES-OF-DELETE-ACTION9 =
             (DELETION12 (DOMINATED BY FILE-DELETION)
              (STATE-CHANGE-OBJECT8 = (FILE9 (DOMINATED BY FILE)))))))))))))))
```

Interpretation 2 (TELL-ACTION23):

```
(HAS-GOAL39 (DOMAIN USER)
 (RANGE
  (TELL-ACTION23 (DOMINATED BY ASK-DESCRIPTION-QUESTION) (ACTOR35 = USER)
   (HEARER21 = UNIX-CONSULTANT) (SPEAKER20 = USER)
   (UTTERANCE21 =
    (DESCRIPTION-QUESTION0 (DOMINATED BY DESCRIPTION-QUESTION)
     (WHAT-IS17 =
      (THING2 (DOMINATED BY THING)
       (PLANFOR50* =
         (PLAN75 (DOMINATED BY PLAN)
          (PLAN-STEP68 =
           (DELETE-ACTION7 (DOMINATED BY DELETE-ACTION)
            (CAUSES-OF-DELETE-ACTION10 =
             (DELETION13 (DOMINATED BY FILE-DELETION)
              (STATE-CHANGE-OBJECT9 = (FILE10 (DOMINATED BY FILE)))))))))))))))
```

6.4.3. Detecting Topic Ambiguity

Topic ambiguity is detected during semantic interpretation. Topic ambiguity can be checked for in any utterance, but *must* be checked for in at least verification questions, specification questions, and negations. When one of these concepts is encountered, the semantic interpreter builds one representation for each of the potential topics of that concept. In the absence of information about the intonation of the utterance, the semantic interpreter can only make intelligent guesses about which concepts might be appropriate topics. There is a tradeoff here between selecting a large number of potential topics so as to ensure that the intended topic is included, and selecting as few potential topics as possible so as to reduce the amount of work that will be required to resolve topic ambiguities. At the very least, when the apparent topic (that is, the root concept produced by the semantic interpreter to represent the content of the utterance) is a relation, the semantic interpreter should select each of the concepts that are connected by that relation as potential topics. For example, suppose the utterance being analyzed is:

User: Does mv delete files?

In this example, the apparent topic is the relation between the **mv** command and file deletion (since the subject is **mv** and the predicate is file deletion). Therefore, each of these concepts should be considered as potential topics of the verification question.

Processing Example

This trace demonstrates the detection of topic ambiguity. It shows the interpretations of the utterance:

User: Does rogue[3] delete files?

that are given as input to **PAGAN**. Notice that only two potential topics are suggested by the semantic interpreter: rogue, and deletion. A more cautious semantic interpretation algorithm would include the direct object in the set of potential topics, at the expense of introducing yet another ambiguity.

User: Does rogue delete files?

This trace shows the two interpretations of this utterance produced by the semantic interpreter.

The parser produced the following interpretations:

[3] Rogue is a game program. One version has the property that it deletes files that the player indicates to contain saved game states, whether or not those files do contain such saved states.

The topic of the utterance in the first interpretation is the range of **WHAT-IS11**, *which is* **ROGUE**.

Interpretation 1 (TELL-ACTION13):

```
(HAS-GOAL15 (DOMAIN USER)
 (RANGE
  (TELL-ACTION13 (DOMINATED BY ASK-VERIFICATION-QUESTION)
   (ACTOR16 = USER) (HEARER10 = UNIX-CONSULTANT) (SPEAKER9 = USER)
   (UTTERANCE10 =
    (VERIFICATION-QUESTION2 (DOMINATED BY VERIFICATION-QUESTION)
     (WHAT-IS11 =
      (ROGUE
       (ACTOR15* =
          (DELETE-ACTION9 (DOMINATED BY DELETE-ACTION)
           (CAUSES-OF-DELETE-ACTION10 =
            (DELETION11 (DOMINATED BY DELETION)
             (STATE-CHANGE-OBJECT10 = (FILES2 (DOMINATED BY FILES)))))))))))))
```

The topic of the utterance in the second interpretation is the range of **WHAT-IS12**, *which is a* **DELETE-ACTION**. *Notice that the only difference between this representation and the representation shown above is the concept to which the* **WHAT-IS** *of the question is attached.*

Interpretation 2 (TELL-ACTION14):

```
(HAS-GOAL16 (DOMAIN USER)
 (RANGE
  (TELL-ACTION14 (DOMINATED BY ASK-VERIFICATION-QUESTION)
   (ACTOR18 = USER) (HEARER11 = UNIX-CONSULTANT) (SPEAKER10 = USER)
   (UTTERANCE11 =
    (VERIFICATION-QUESTION3 (DOMINATED BY VERIFICATION-QUESTION)
     (WHAT-IS12 =
      (DELETE-ACTION10 (DOMINATED BY DELETE-ACTION) (ACTOR17 = ROGUE)
       (CAUSES-OF-DELETE-ACTION11 =
          (DELETION12 (DOMINATED BY DELETION)
           (STATE-CHANGE-OBJECT11 = (FILES3 (DOMINATED BY FILES)))))))))))))
```

6.4.4. Detecting Goal Ambiguity

As described in Chapter 5, planfor explanations for an action are found by looking upward in the knowledge hierarchy for applicable planfors. Goal ambiguity is detected when this search process discovers more than one applicable planfor. Both utterance-level and domain-level goal ambiguities are detected in this way. An example of a pair of planfors that lead to the detection of direct/indirect ambiguity is shown in Figure 6.2.

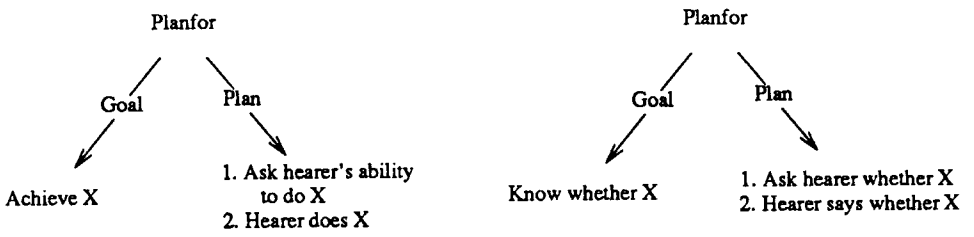

Figure 6.2. Planfors That Lead To Goal Ambiguity

This pair of planfors is the source of direct/indirect goal ambiguity. The ambiguity arises because the first plan step in each plan can match the same utterance. For example, the question "Can you tell me how to mail a letter?" will match the first step of either of these planfors.

Processing Example

The following trace shows PAGAN's initial processing of the question:

User: Can you tell me how to edit a file?

PAGAN detects two goals that might have led the user to produce the utterance; these goals reflect the direct and the indirect interpretations of the utterance.

User: Can you tell me how to edit a file?

The parser produces a single interpretation of the user's question.

The parser produced the following interpretation:

```
Interpretation 1 (TELL-ACTION59):
(HAS-GOAL129 (DOMAIN USER)
 (RANGE
  (TELL-ACTION59 (DOMINATED BY ASK-ABILITY-QUESTION) (ACTOR71 = USER)
   (HEARER50 = UNIX-CONSULTANT) (SPEAKER45 = USER)
   (UTTERANCE52 =
    (ABILITY-QUESTION4 (DOMINATED BY ABILITY-QUESTION)
     (WHAT-IS38 =
      (TELL-ACTION58 (DOMINATED BY TELL-ACTION)
       (ACTOR70 = UNIX-CONSULTANT) (HAS-ABILITY4* = UNIX-CONSULTANT)
       (HEARER49 = USER)
       (UTTERANCE51 =
          (THING18 (DOMINATED BY THING)
           (PLANFOR105* =
            (PLAN127 (DOMINATED BY PLAN)
             (PLAN-STEP138 =
```

(EDIT-ACTION5 (DOMINATED BY EDIT-ACTION)
(CAUSES-OF-EDIT-ACTION6 =
(EDITING7 (DOMINATED BY EDITING)
(STATE-CHANGE-OBJECT24 = (FILE29 (DOMINATED BY FILE))))))))))))))))))

Before searching for planfor explanations of the utterance, PAGAN *first checks to see that the interpretation of the utterance is plausible, and that it does not immediately match an expectation.*

Attempting to resolve any ambiguities in TELL-ACTION59
Retrieving potential conflicts with TELL-ACTION59 and its plan.
No conflict found.

Attempting to find an explanation for TELL-ACTION59.
Determining whether TELL-ACTION59 is grounded in the preceding dialogue.
TELL-ACTION59 was not grounded in the preceding dialogue.

Trying to find planfor explanations for TELL-ACTION59.

In looking for an explanation of the user's utterance, PAGAN *finds two applicable planfors. The first,* ASK-POLITELY-PLANFOR, *states that a way to ask a hearer to do something is to ask the hearer's ability to do it. This planfor represents the indirect interpretation of the utterance.*

Found ASK-POLITELY-PLANFOR as an explanation for TELL-ACTION59.
Synopsis:
Goal: request hearer do something;
Step 1: Ask hearer's ability to do it.
Inferred goal is TELL-ACTION61.

The second applicable planfor, PLANFOR2, *states that in order to know whether something is true, one should ask whether it is true. This planfor represents the direct interpretation of the utterance.*

Found PLANFOR2 as an explanation for TELL-ACTION59.
Synopsis:
Goal: Know whether f(X) = Y;
Step 1: Ask whether f(X) = Y;
Step 2: Hearer says yes or no.
Inferred goal is KNOWING-WHETHER25.

Before continuing, PAGAN *must determine whether these two interpretations are compatible with one another. They are not, so* PAGAN *marks them as such. If one of the interpretations is subsequently selected, the other will be rejected (and vice versa).*

Determining whether ASK-POLITELY-PLANFOR and PLANFOR2 are incompatible.
The explanations are incompatible.
Explanation found for TELL-ACTION59.

6.4.5. Determining Whether Two Interpretations are Incompatible

When multiple explanation chains for an utterance that are mutually compatible are encountered, the correct action for PAGAN is to accept the existence of multiple explanations of the utterance. That is, each of the explanation chains should be part of the final explanation of the utterance. It is by allowing more than one explanation chain to apply to a single utterance in this way that the algorithm generates explanations that satisfy the breadth completeness criterion. The philosophy here is that if all reasonable explanations for an utterance are first proposed, then all those that are incorrect are thrown out, the correct, complete explanation of the utterance will remain.

Given two or more explanations of an utterance, a goal analyzer must have the ability to determine whether the explanations are compatible with one another, or whether one of the interpretations must be selected as the correct one at the expense of the others. Goal ambiguity is treated differently from other types of ambiguity in this regard. When the ambiguity is sentence or utterance ambiguity, the alternatives are always considered to be mutually incompatible. While this approach fails to illuminate some subtleties of language (such as inferring both interpretations of a pun), it handles most phenomena encountered in straightforward cooperative dialogue quite well.

When the ambiguity is goal ambiguity, the principle of horizontal completeness provides a good heuristic for determining when the explanations are compatible. Those explanations that explain exactly the same components of the utterance are taken to be incompatible; those that explain different portions of the utterance are compatible. That is, if the two potential planfor explanations are attached to the same concept in the knowledge hierarchy, or one is attached to a concept that is the ancestor of the concept the other planfor is attached to, then the two explanations are incompatible. Otherwise, the two explanations are explaining different portions of the utterance, and so are compatible. For example, the planfors shown in Figure 6.2 are incompatible, because the first steps of their plans are hierarchically related.

6.5. Summary

There are three broad classes of ambiguity that arise in dialogue. *Sentence ambiguity* occurs when a sentence is ambiguous outside of any particular context. Lexical ambiguity and structural ambiguity are both sentence ambiguities. *Utterance ambiguity* occurs when the use of a sentence is ambiguous in a particular context. There are three types of utterance ambiguity: referential ambiguity, *categorization ambiguity* (ambiguity arising from the existence of multiple plausible subcategories for a concept in a particular category), and *topic ambiguity* (ambiguity arising when there is more than one potential primary focus of an utterance). Finally, *goal ambiguity* occurs when a particular action (such as an utterance), while it is unambiguous relative to the two preceding categories, nevertheless can be used for more than one purpose.

While these types of ambiguity may sometimes be handled by other mechanisms, there are also times when each of them will have to be addressed by the goal analyzer. The next chapter shows how a single algorithm can be used to handle each of these types of ambiguity during goal analysis.

Chapter 7

Handling Ambiguity

When ambiguity is detected and it cannot be resolved by other components, it must be addressed by PAGAN. This chapter describes the algorithm PAGAN uses to handle ambiguity. It is important to note that no matter what the *source* of an ambiguity, the same algorithm is used to handle it.

There are two ways to handle ambiguity:

1. Resolve the ambiguity, and continue processing
2. Ignore the ambiguity, processing each alternative individually

Ambiguity resolution can be achieved in either of two ways:

1. By rejecting all the incorrect interpretations, leaving the correct one
2. By selecting the correct interpretation outright

Rejection of an interpretation is done by detecting a conflict between the interpretation and some other knowledge held by the system. Such a conflict is called a *knowledge conflict*. Selection of an interpretation is done either by matching the interpretation to an expectation (as described in Chapter 5), or by naturally preferring that interpretation over others.

A tree showing a taxonomy of these approaches to handling ambiguity is shown in Figure 7.1. The first part of this chapter is organized around a pre-order traversal of this tree. After this discussion of how to handle ambiguity, I comment upon what the system as a whole should do if it cannot resolve an ambiguity. Finally, I discuss how the techniques described in this chapter can be implemented.

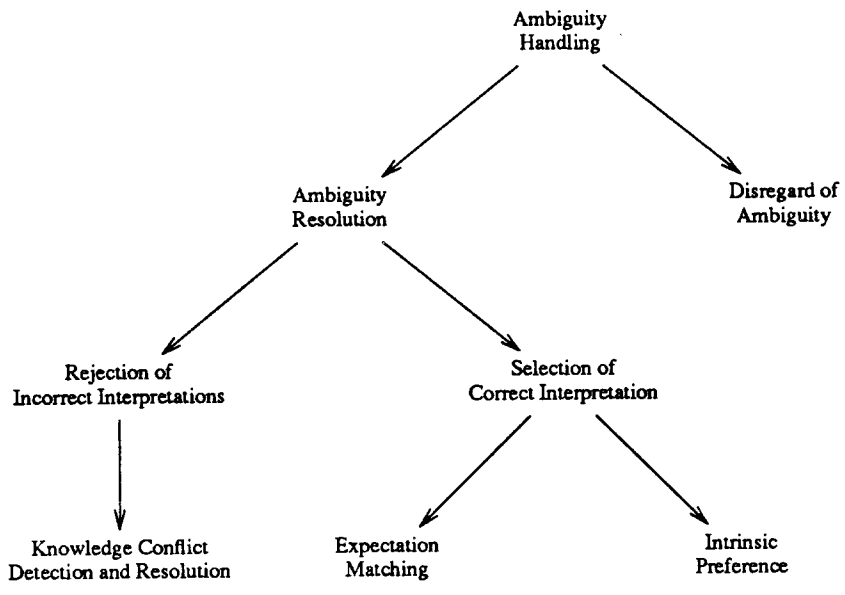

Figure 7.1. A Taxonomy of Methods for Handling Ambiguity

7.1. Ambiguity Resolution

The best approach to handling ambiguity is to resolve the ambiguity. This means that one of the interpretations is taken to be correct, at the expense of the other interpretations. Ambiguities arise because sentences themselves do not contain enough information to reveal the intentions behind their use. Therefore, to resolve ambiguity, knowledge from sources other than the sentence in question must be used.

There are two rules for determining how to resolve an ambiguity. These rules are valid not just for goal analysis, but for any interpretation task that encounters multiple interpretations. The first of these rules concerns how an explanation relates to the system's knowledge of the world. The second rule concerns how an explanation relates to the context in which the action to be explained was performed:

> **Rule 1**: Reject an interpretation if that interpretation is in conflict with some belief of the system, and if confidence in the interpretation is less than confidence in the belief.
>
> **Rule 2**: Select an interpretation if context indicates there is a reason to prefer it over other interpretations.

In the following subsections, I describe these rules in more detail. First, I discuss how a contradiction between an explanation and a system belief can lead to rejection of the explanation. Following this discussion of rejection of an interpretation, I discuss selection of an interpretation as a way to resolve ambiguity. In the next section, I present disregard of ambiguity as an alternative to knowledge-based ambiguity resolution.

7.1.1. Rejecting an Interpretation

Rejection of an interpretation of an utterance is the first of two methods for resolving an ambiguity. Rejection of an interpretation hinges on whether the knowledge contained in or inferred from the interpretation conflicts with beliefs held by the system. Such conflicts are called *knowledge conflicts*. If a knowledge conflict is found, and if the mechanism that makes inferences about the interpretation works properly, then one of two conditions holds: either the interpretation is incorrect; or the system's belief is incorrect. If the system determines that the interpretation of the utterance is incorrect, then the interpretation can be rejected as a plausible explanation of the utterance. Rejecting the interpretation will either resolve the ambiguity, or at the very least reduce the number of interpretations under consideration. On the other hand, if the system decides that the interpretation of the utterance is more likely to be correct than the conflicting belief, then the belief will be abandoned.

If rejection is to be used to resolve ambiguity, two new capabilities are required. First, **PAGAN** needs the ability to detect situations in which an explanation is in conflict with a previously held belief. Secondly, **PAGAN** needs a method for weighing the strength of its confidence in the explanation and in the belief. In the following subsections, I first show how a belief held by a model of the user can conflict with a belief about the user inferred from an explanation of an utterance. I then discuss how such conflicts can be detected. Finally, I describe how these conflicts can be resolved, and how such resolution can lead to the rejection of an interpretation of an utterance.

7.1.1.1. The User Model as Source of Knowledge Conflicts

The main type of knowledge that may conflict with knowledge gleaned from an utterance is knowledge of the speaker of the utterance. Such knowledge is accessed by referring to a user modeling component, such as **KNOME** [Chin, 1988]. The user model is a component of a dialogue system that maintains long-term knowledge about the other dialogue participants.[1] Such knowledge may include for example knowledge of the user's knowledge, knowledge of the user's beliefs, knowledge of the user's goals, and knowledge of the user's capabilities. Any of these types of knowledge maintained by the user model may conflict with a portion of an explanation of an utterance. The ways in which a portion of an explanation may conflict with such knowledge is the subject of the following sections.

[1] The meaning of 'long-term' here is relative to the duration of an utterance; I do not mean to imply that this knowledge is necessarily retained between dialogues with the same user.

7.1.1.2. Sources of Knowledge Conflicts Within an Explanation

An explanation of an utterance has many components, any one of which might be the source of a knowledge conflict. It is useful though to divide the sources of conflicts within an explanation into three categories:

1. Conflicts with the inferred goal
2. Conflicts with the inferred plan
3. Conflicts with the explanation as a whole

The first category encompasses those components of an explanation most closely associated with the goal of the planfor; the second category includes those components most closely associated with one of the steps of the plan; and the third category covers the explanation as a whole. Of course, no component of an explanation will be exclusively in the domain of the goal or of the plan of the explanation; rather, each component is connected to both the plan and to the goal by some chain of relations. For example, the portion of an explanation of a question that represents the concept 'how to delete a file' may be connected to both the goal of the planfor ('speaker wants to know how to delete a file'), to the first step of the plan ('speaker asks how to delete a file'), and to the second step of the plan ('speaker is told how to delete a file'). Although it may be impossible to associate a concept with only one of the categories listed above, it is nevertheless useful to assign each concept to one or more of the categories, and to use the categories to study knowledge conflicts.

I have found six kinds of knowledge conflict that can lead to the resolution of ambiguity:

1. Goal conflicts
2. Goal obtains
3. Knowledge is deficient
4. Unable to perform action
5. Consequent conflicts
6. Better plan known

These types of knowledge conflict are discussed in the following sections.

7.1.1.2.1. Conflicts with the Goal of an Explanation

The goal of an explanation may conflict with the model of the user in two ways:

1. The goal itself conflicts
2. The knowledge that would be required to hold the goal conflicts

The goal of an explanation conflicts with the model of the user when the user model indicates that the user does not hold that goal. There are two reasons that the user model might indicate such a conflict:

1. The user would not want the goal state to hold
2. The user already believes the goal to obtain

The first reason that the user model might indicate a conflict with the goal of an explanation is that it believes that the speaker would not want that goal state to hold:

User: Will an infinite loop crash UNIX?

One interpretation of this question is that the speaker wants to crash UNIX. If the user model held the quite reasonable belief that the user would not want to crash the system, that belief would conflict with the goal of this interpretation. The conflict could be used to resolve the apparent ambiguity in the question. Of course, it is unlikely that the *goal* of not crashing the system would be part of a user model. It is more likely that the user model would contain some sort of preservation theme that would give rise to goals such as this.

The second reason that the user model might indicate a conflict with the goal of an explanation as a whole is that it believes that the speaker already believes the goal to obtain. This rule is derivative of Allen's 'Effects True' heuristic for rating an explanation [1979]. Consider the question:

User: Do you know how to copy a file to a Sun?

When UC is playing the role of consultant to a novice UNIX user, the user model will indicate that the user knows that UC does in fact have that information available. Thus, the direct interpretation of this question (that it is a question about UC's knowledge state) will conflict with this belief of the user model, and will lead to a resolution of the ambiguity. If on the other hand the user model indicates that it believes that the user may have some doubt as to the system's capabilities (such as when being taught new information by an expert), the direct interpretation of this question will not conflict in this way. In this case though, the user model may assume that the speaker (i.e. the person adding knowledge to UC) already knows how to copy a file to a Sun. If the indirect interpretation of the utterance (that it is a request to be told how to copy a file) is under consideration, it will conflict with this belief of the user model. This conflict will allow the ambiguity to be resolved.

A goal may also give rise to a knowledge conflict when the user model does not believe that the speaker has all the knowledge that would be required to hold the goal. For example, suppose a novice UNIX user asks:

User: How can I delete a file?

One explanation of this utterance includes the goal of freeing the disk blocks associated with the file. However, the user model should know that a novice is unlikely even to understand the concept of disk blocks, much less want to know how to manipulate them. This interpretation then will lead to a knowledge conflict with the user model that can be used to reject the interpretation. Note that the topic of a description question must not be ruled out in this way, since such a question does not presuppose that the speaker understands the referenced concept.

Conversely, an explanation of an utterance may imply that the speaker does not possess certain knowledge, where the user model indicates that the speaker does have that knowledge:

User: What is a good text processor on this system?

This question exhibits categorization ambiguity. One explanation of the question is that the speaker wants to know what it means to be a good text processor on this system. This interpretation takes part in a knowledge conflict if the user model indicates that the speaker does know the meaning of the concept about which she has inquired.

7.1.1.2.2. Conflicts with the Plan of an Explanation

Because plans are usually composed of actions taken by one or more agents, conflicts may arise from two aspects of a plan:

1. The abilities of the agent(s)
2. The consequences of the action(s)

First, a plan may require that an agent perform a particular action, while the user model indicates that the speaker believes that the agent cannot perform that action. This rule is derivative of Allen's 'Preconditions False' heuristic for rating an explanation [1979]. Consider the question:

User: Do you know how to remove my file named dead.letter?

Suppose that PAGAN is considering the interpretation that the user wants UC to delete the file. If the user model believes that the user is aware that UC is only able to give advice and cannot itself take action in the UNIX domain, then there is a knowledge conflict between the plan portion of the explanation and the user model's knowledge. The conflict will be useful in rejecting this interpretation of the question.

Secondly, one of the consequences of an action that is a part of the plan may give rise to a knowledge conflict. Three criteria must hold true for such a conflict to arise:

1. The speaker must believe that the consequent follows
2. The speaker must desire that the consequent not hold
3. The consequent must not be the topic of conversation

For example, suppose the user asks:

User: Do you know how to delete a file?

One interpretation of this utterance is the literal one, which says that the user is actually questioning UC's abilities. One of the effects of this plan is that the hearer may be offended if the answer to the question is obviously affirmative. If the user model indicates that the user knows that such a question might be taken as offensive, and indicates furthermore that the user does not desire to offend UC, then a knowledge conflict stems from this plan.

The third criterion, that the undesirable effect not be the topic of conversation, is needed to allow proper processing of utterances that ask how to avoid undesirable effects:

User: How can I run uncompress on my file without deleting it?

Here, the speaker is asking how the uncompress plan can be altered so that it does not delete the file in question. It would be incorrect in this example to reject the interpretation that the speaker wants to use the uncompress plan because that plan has an undesired effect.

7.1.1.2.3. Conflicts with the Entirety of an Explanation

The conflicts I have discussed so far have been conflicts between the user model and *portions* of the explanation of the utterance. It is also possible that a belief held by the user model may conflict with the entirety of the explanation. Consider the question:

User: Does rogue delete files?

This question exhibits topic ambiguity. One of its interpretations is that the user wants to know how to delete files. However, if the user model indicates that the speaker believes both that rm is the program specifically designed to delete files, and that rogue is designed as a game, then a conflict arises between this fact and the interpretation that the speaker wants to know another way to delete files. Notice that this is not merely a conflict with the goal of the explanation. If the speaker only knew a very bad plan for deleting files, then it would be perfectly reasonable for her to seek a better method. It is only the fact that a better plan for the goal is already known by the speaker that causes the conflict. Thus, the conflict is with the explanation as a whole. This points out an advantage of the planfor representation; statements need not be restricted to plans or goals alone; they may be predicated about the planfor as a whole.

7.1.1.3. Resolving Knowledge Conflicts

Knowledge conflicts are resolved by selecting that element of the conflict in which the most confidence is held, and rejecting the other. If the selected element is derived from the user model, the conflicting explanation is rejected, possibly resolving the ambiguity. If however the selected element is derived from the interpretation of the utterance, the user model is deemed to have been incorrect. In this case, not only must the user model be updated to reflect the changed belief, but another method must be sought to resolve the ambiguity. If neither element of the conflict inspires significantly more confidence than its competitor, then nothing significant is learned from the conflict, and another method must be used to resolve the ambiguity.

7.1.2. Selecting an Interpretation

If there are several explanations of an utterance, and none is in conflict with the system's beliefs, it may still be possible to select among them. If there is some reason to prefer one of the explanations over the others, then that explanation may be *selected* as the correct explanation. There are two reasons that one explanation might be preferred over another:

1. Expectation matching
2. Intrinsic preference

Knowledge of previous dialogue is accessed by **PAGAN** through the intentional model of the dialogue it builds and maintains. Chapter 5 discussed expectation matching as a way to build explanations that meet the grounding criterion. Expectation matching is also useful as a way to resolve ambiguities, by indicating that an interpretation that matches an expectation should be selected as the correct interpretation. To see how this kind of selection of a competing planfor explanation can be used to resolve an ambiguity, suppose that **PAGAN** is analyzing the user's utterance in this exchange:

UC: Is the file in your home directory?
User: It's in /tmp.

If the semantic interpreter is unable to determine whether 'it' refers to the file or the home directory, the response will exhibit utterance ambiguity (because each of the semantic interpretations is a candidate for goal analysis). The grounding of the response in UC's goal of knowing the location of the file will resolve this ambiguity.

Secondly, preference can be given to an interpretation if the goals of that interpretation are more likely to hold than those of competing interpretations. For example, a person is more likely to want to delete a file than a directory. If **PAGAN** is trying to decide between a file and a directory as the referent of 'it' in the question:

User: How can I delete it?

then this knowledge can be used to select the file as the referent.

7.2. Disregard of Ambiguity

The alternative to resolving ambiguity is ignoring it. There are three cases in which ignoring ambiguity can lead to acceptable results:

1. When the ambiguity can be resolved at a later point
2. When the system does not require a single interpretation
3. When the ambiguity is real ambiguity

The first way that disregard of ambiguity can be useful is as a delaying tactic. By carrying forward multiple interpretations of an input, **PAGAN** may *eventually* be able to apply the principles discussed above to resolve the ambiguity. For example, consider the sequence:

User: I want more disk space.
User: Does mv delete files?

This last question exhibits topic ambiguity; it can either be a question about what mv does, or a question about how to delete files. Knowing only this, the ambiguity cannot be resolved. However, if the ambiguity is temporarily ignored and both interpretations are pursued, the latter interpretation will ultimately be selected. This is because deleting files is a way to increase disk space, and therefore matches a goal introduced by the preceding statement. The **Lucy** system [Rich et al., 1987] carries this type of delay a step further. In **Lucy**, rival explanations of an utterance are not even constructed until there is some hope of resolving the ambiguity.

Secondly, in some cases, ambiguity may be ignored altogether. When **UC** can easily address the goal of each competing interpretation, then such disregard is acceptable. For example, suppose that **PAGAN** is unable to determine whether:

User: Do you know where the lisp interpreter is located?

is intended as a direct or an indirect request. Since both interpretations can be addressed by a response such as:

UC: Yes, it's in /usr/ucb/lisp.

it is possible in this example for **PAGAN** to ignore the ambiguity. Doing so of course shifts the burden of coping with the ambiguity onto the planner and the expression mechanism, which must engineer a response such as this one.

Finally, in rare cases, a perceived ambiguity is a real ambiguity. In such cases, even a human observer cannot decide which of the interpretations should be adopted. If human performance is taken as a limit on the performance of a computer program, then disregard of real ambiguity by PAGAN is acceptable.

7.3. Unresolved Ambiguity

If ambiguity remains after execution of this algorithm, PAGAN can do nothing more to resolve it; it is up to the rest of the system to handle the ambiguity. There are two possible causes of this situation:

1. The ambiguity is real ambiguity that the system must confront
2. The ambiguity is resolvable ambiguity that PAGAN failed to resolve

The first possibility is that the unresolved ambiguity is a real ambiguity that the system must confront. That is, the speaker truly failed to make her intentions clear. The second possibility is that there were sufficient clues provided by the speaker to disambiguate the utterance, but for some reason PAGAN was unable to do so. Since without further knowledge sources there is no way for the system to distinguish between these two possibilities, it should assume that the ambiguity is real. The user will be quick to point out the error if the ambiguity was resolvable.

There are three principle tactics that a system may adopt when confronted with unresolved ambiguity:

1. Query the user
2. Arbitrarily select one of the interpretations
3. Finesse the ambiguity

The first option the system has is to ask the user which interpretation was intended:

User: How can I save my program?
UC: Are you trying to save the source code or the executable code?

Here, PAGAN has not been able to resolve the lexical ambiguity in the word 'program.' The system has chosen to deal with the ambiguity by explicitly asking the speaker which interpretation was intended.

The second option at the system's disposal is to select one of the interpretations arbitrarily, and treat it as if it were the correct one:

User: How can I see my file?
UC: Use 'more.'

In this example, 'seeing' a file could mean either viewing it on a screen, or having a print-out of the file to peruse. Lacking any simple way to determine which outcome the speaker desired, the system has simply chosen one of the interpretations arbitrarily, and told the user how to achieve it. The expectation is that if the choice was incorrect, the user will alert the system to that fact.

Finally, the system can finesse the ambiguity by addressing each of the goals, or by addressing none of them:

> User: Can you tell me how to delete a directory?
> UC: Yes, you use rmdir.
>
> ---
>
> User: How can I copy cs374118's program?
> UC: I can't tell you how to steal another student's work.

In the first example, the speaker has made either a direct or an indirect request. The system is able to proceed smoothly without knowing which of the interpretations was intended, by responding with an utterance that satisfies both goals. In the second example, the ambiguity introduced by the word 'program' is ignored altogether; the system rejects both goals as inappropriate for this user, and informs the user of its rejection.

7.4. Implementation

The bulk of the processing power used in handling ambiguity is directed toward the detection and resolution of knowledge conflicts. The next section discusses how this is achieved. The following sections cover selection of an interpretation, and disregard of ambiguity.

7.4.1. Detecting and Resolving Knowledge Conflicts

The detection of a knowledge conflict is a three-step process:

1. Selection of a knowledge conflict type
2. Selection of a component of the explanation
3. Querying of the user model

The first step in detecting a knowledge conflict is to select one of the knowledge conflict types presented in the preceding sections:

1. Goal conflicts
2. Goal obtains
3. Knowledge is deficient
4. Unable to perform action
5. Consequent conflicts
6. Better plan known

Once a knowledge conflict type has been chosen, the second step is to select some component of the explanation to examine for this type of conflict. To reduce the number of queries to the user model, only those components of the explanation that are appropriate for the chosen knowledge conflict type should be considered. Delving deep within the network that represents the explanation should be avoided for the same reason. Therefore, the following rules are used in selecting components of the explanation to be checked for knowledge conflicts. Each potential conflict type is examined every time a knowledge conflict is sought:

Goal conflicts: Select only the motivating goal.
Goal obtains: Select only the motivating goal.
Knowledge is deficient: Select only components of the goal that relate directly to the domain of discourse, and that are within one relation of the motivating goal. This rule handles most cases where knowledge deficiency is applicable, without increasing the computational load significantly.
Unable to perform action: Select only top-level plan steps that are actions.
Consequent conflicts: Select only major effects of plan steps that are events.
Better plan known: Select only the planfor used to generate the explanation.

Once the knowledge conflict type and explanation component have been selected, the final step in knowledge conflict detection is to query the user model as to whether it holds any beliefs about the user that conflict with the selected component of the explanation in the suggested way. The user model must not only select the beliefs it holds that conflict with the selected component, but must also rate its confidence in those beliefs. Currently, a simplified version of Chin's KNOME user model [1988], adapted to PAGAN's requirements, is used for this task. It treats the concept provided by PAGAN as an assertion, and determines whether it is likely that the user believes that assertion. Note that a truth maintenance system [Doyle 1979] could be used to mediate between PAGAN and the user model in knowledge conflicts. However, the selection of beliefs to reject when a conflict arises must be directed by PAGAN and the user model, not by the truth maintenance system. PAGAN does not currently use such a system.

A simple way to determine how confident PAGAN should be about a particular interpretation of an utterance is to heuristically map from an explanation to a rating drawn from the set {VERY-SURE, SURE, NOT-SURE}. The following heuristics are used to perform the mapping:

Heuristic 1: If the explanation matches an expectation, then the system is VERY-SURE of the interpretation.

Heuristic 2: If the explanation is the only one available, then the system is SURE of the interpretation.

Heuristic 3: If the explanation is one of many that the system is considering, then the system is NOT-SURE of the interpretation.

While this set of certainty levels and these heuristics are simplistic, they do nevertheless provide a reasonable first approach to the problem of assigning confidence levels to explanations.

Processing Example

The following trace shows PAGAN's processing of the question:

User: Will an infinite loop crash UNIX?

This utterance exhibits topic ambiguity; the user may want to know the effects of an infinite loop, or how to crash UNIX. The ambiguity is resolved through a knowledge conflict with the user model.

User: Will an infinite loop crash UNIX?

The parser produced the following interpretations:

Two interpretations of the user's utterance are produced by the semantic interpreter. The topic of the first (the value of **WHAT-IS55***) is an infinite loop, while the topic of the second (the value of* **WHAT-IS56***) is a UNIX crash.*

```
Interpretation 1 (TELL-ACTION77):
(HAS-GOAL172 (DOMAIN USER)
 (RANGE
  (TELL-ACTION77 (DOMINATED BY ASK-VERIFICATION-QUESTION)
   (ACTOR92 = USER) (HEARER66 = UNIX-CONSULTANT) (SPEAKER61 = USER)
   (UTTERANCE68 =
    (VERIFICATION-QUESTION18 (DOMINATED BY VERIFICATION-QUESTION)
     (WHAT-IS55 =
      (INFINITE-LOOP13 (DOMINATED BY INFINITE-LOOP12)
       (DOMINATED BY ANIMATE)
       (ACTOR91* =
         (CRASH-ACTION6 (DOMINATED BY CRASH-ACTION)
          (CAUSES-OF-CRASH-ACTION9 =
           (CRASH12 (DOMINATED BY CRASH)
            (STATE-CHANGE-OBJECT-OF-CRASH6 = UNIX)))))))))))
```

Interpretation 2 (TELL-ACTION78):
(HAS-GOAL173 (DOMAIN USER)
(RANGE
(TELL-ACTION78 (DOMINATED BY ASK-VERIFICATION-QUESTION)
(ACTOR94 = USER) (HEARER67 = UNIX-CONSULTANT) (SPEAKER62 = USER)
(UTTERANCE69 =
(VERIFICATION-QUESTION19 (DOMINATED BY VERIFICATION-QUESTION)
(WHAT-IS56 =
(CRASH-ACTION7 (DOMINATED BY CRASH-ACTION)
(ACTOR93 =
(INFINITE-LOOP15 (DOMINATED BY INFINITE-LOOP14)
(DOMINATED BY ANIMATE)))
(CAUSES-OF-CRASH-ACTION10 =
(CRASH13 (DOMINATED BY CRASH)
(STATE-CHANGE-OBJECT-OF-CRASH7 = UNIX)))))))))

The ambiguity cannot be resolved at this point.

Attempting to resolve any ambiguities in TELL-ACTION78
Retrieving potential conflicts with TELL-ACTION78 and its plan.
No conflict found.

Attempting to resolve any ambiguities in TELL-ACTION77
Retrieving potential conflicts with TELL-ACTION77 and its plan.
No conflict found.

PAGAN *searches for an explanation for each of the interpretations. The same planfor is found to explain them both. This planfor states that a plan for knowing whether a condition holds is to ask whether it holds, and to receive an answer.*

Attempting to find an explanation for TELL-ACTION77.
Determining whether TELL-ACTION77 is grounded in the preceding dialogue.
TELL-ACTION77 was not grounded in the preceding dialogue.

Trying to find planfor explanations for TELL-ACTION77.
Found PLANFOR138 as an explanation for TELL-ACTION77.
Synopsis:
Goal: Know whether f(X) = Y;
Step 1: Ask whether f(X) = Y;
Step 2: Hearer says yes or no.
Inferred goal is KNOWING-WHETHER28.

Explanation found for TELL-ACTION77.

Attempting to find an explanation for TELL-ACTION78.
Determining whether TELL-ACTION78 is grounded in the preceding dialogue.
TELL-ACTION78 was not grounded in the preceding dialogue.

Trying to find planfor explanations for TELL-ACTION78.
Found PLANFOR138 as an explanation for TELL-ACTION78.
Synopsis:
Goal: Know whether f(X) = Y;
Step 1: Ask whether f(X) = Y;
Step 2: Hearer says yes or no.
Inferred goal is KNOWING-WHETHER29.

Explanation found for TELL-ACTION78.

Again, the ambiguity cannot be resolved.

Attempting to resolve any ambiguities in KNOWING-WHETHER29
Retrieving potential conflicts with KNOWING-WHETHER29 and its plan.
No conflict found.

Attempting to resolve any ambiguities in KNOWING-WHETHER28
Retrieving potential conflicts with KNOWING-WHETHER28 and its plan.
No conflict found.

The parallel development of the two explanations continues with the use of a single planfor applicable to both. This planfor states that a plan for the goal of knowing something is to know that it has a particular value.

Attempting to find an explanation for KNOWING-WHETHER28.
Determining whether KNOWING-WHETHER28 is grounded in the preceding dialogue.
KNOWING-WHETHER28 was not grounded in the preceding dialogue.

Trying to find planfor explanations for KNOWING-WHETHER28.
Found PLANFOR137 as an explanation for KNOWING-WHETHER28.
Synopsis:
Goal: Know f(X);
Step 1: Know that f(X) = Y
Inferred goal is KNOWING-THAT33.

Explanation found for KNOWING-WHETHER28.

Attempting to find an explanation for KNOWING-WHETHER29.
Determining whether KNOWING-WHETHER29 is grounded in the preceding dialogue.
KNOWING-WHETHER29 was not grounded in the preceding dialogue.

Trying to find planfor explanations for KNOWING-WHETHER29.
Found PLANFOR137 as an explanation for KNOWING-WHETHER29.
Synopsis:
Goal: Know f(X);
Step 1: Know that f(X) = Y
Inferred goal is KNOWING-THAT34.

Explanation found for KNOWING-WHETHER29.

The ambiguity still cannot be resolved.

Attempting to resolve any ambiguities in KNOWING-THAT34
Retrieving potential conflicts with KNOWING-THAT34 and its plan.
No conflict found.

Attempting to resolve any ambiguities in KNOWING-THAT33
Retrieving potential conflicts with KNOWING-THAT33 and its plan.
No conflict found.

PAGAN *tries to find an explanation for the goal of knowing the outcome of an infinite loop. No such explanation is found.*

Attempting to find an explanation for KNOWING-THAT33.
Determining whether KNOWING-THAT33 is grounded in the preceding dialogue.
KNOWING-THAT33 was not grounded in the preceding dialogue.

Trying to find planfor explanations for KNOWING-THAT33.
No planfor explanations found for KNOWING-THAT33.

Determining whether KNOWING-THAT33 is explained by a theme.
KNOWING-THAT33 was not explained by a theme.

No explanation found for KNOWING-THAT33.

Exploring the other option, PAGAN *infers that the user may want to know how to crash UNIX in order to actually crash it.*

Attempting to find an explanation for KNOWING-THAT34.
Determining whether KNOWING-THAT34 is grounded in the preceding dialogue.
KNOWING-THAT34 was not grounded in the preceding dialogue.

Trying to find planfor explanations for KNOWING-THAT34.
Found PLANFOR144 as an explanation for KNOWING-THAT34.
Synopsis:
Goal: Achieve X;
Step 1: Know how to X;
Step 2: Do X
Inferred goal is CRASH15.

Explanation found for KNOWING-THAT34.

> PAGAN *now examines the goal of crashing UNIX. This goal is found to conflict with the goals that the user model believes the user holds. Therefore, this interpretation is rejected. The correct interpretation, that the user wants to know the outcome of an infinite loop, remains.*

Attempting to resolve any ambiguities in CRASH15
Retrieving potential conflicts with CRASH15 and its plan.
Conflict found: inferred goal conflicts with user's goals.
Rejecting CRASH15 as an explanation of KNOWING-THAT34.

7.4.2. Selecting an Interpretation

Expectation matching is performed by asserting that the event to be explained lies below the expected event in the knowledge hierarchy. If no error arises in making this assertion, the event to be explained successfully matches the expectation, and therefore the interpretation of which it is a part is selected as the correct one. This expectation matching process was described in detail in the implementation section of Chapter 5.

Intrinsic preference of one interpretation over another is implemented by associating a likelihood rating with each goal type. When two explanations of an utterance are in conflict with one another, the ratings of their goals are compared. If one rating exceeds the other by a predetermined amount, the corresponding explanation is taken as the correct one. However, because intrinsic preference does not take the particular circumstances of an utterance into account, the usefulness of this method of ambiguity resolution is limited to highly constrained domains. Because of this constraint, PAGAN does not currently use intrinsic preference in processing questions about UNIX.

Processing Example

The following trace shows PAGAN's processing of the user's utterance in the exchange:

> UC: Is the file in your directory?
> User: It is in /tmp.

Of note in this trace is the detection of the referential ambiguity of the word 'it,' and its subsequent resolution via grounding in UC's utterance.

UC: Is the file in your directory?

User: It is in /tmp.

The parser produced the following interpretations:

The semantic interpreter is unable to determine whether 'it' refers to the file or the directory, so it produces two interpretations of the user's statement.

Interpretation 1 (TELL-ACTION25):
(HAS-GOAL45 (DOMAIN USER)
 (RANGE
  (TELL-ACTION25 (DOMINATED BY MAKE-STATEMENT) (ACTOR31 = USER)
   (HEARER22 = UNIX-CONSULTANT) (SPEAKER21 = USER)
   (UTTERANCE22 =
    (LOCATION-OF6 (DOMINATED BY LOCATION-OF)
     (LOCATED6 = (FILE16 (DOMINATED BY FILE))) (LOCATION6 = SLASH-TMP))))))

Interpretation 2 (TELL-ACTION26):
(HAS-GOAL46 (DOMAIN USER)
 (RANGE
  (TELL-ACTION26 (DOMINATED BY MAKE-STATEMENT) (ACTOR32 = USER)
   (HEARER23 = UNIX-CONSULTANT) (SPEAKER22 = USER)
   (UTTERANCE23 =
    (LOCATION-OF7 (DOMINATED BY LOCATION-OF)
     (LOCATED7 = (DIRECTORY4 (DOMINATED BY DIRECTORY)))
     (LOCATION7 = SLASH-TMP))))))

No conflict is found with either interpretation, so PAGAN *tries to find an explanation for each of them.*

Attempting to resolve any ambiguities in TELL-ACTION26
  Retrieving potential conflicts with TELL-ACTION26 and its plan.
  No conflict found.

Attempting to resolve any ambiguities in TELL-ACTION25
  Retrieving potential conflicts with TELL-ACTION25 and its plan.
  No conflict found.

Determining whether TELL-ACTION25 is grounded in the preceding dialogue.
Trying to ground TELL-ACTION25 in INDICATE-YES-OR-NO.
  Concreting TELL-ACTION25 to be INDICATE-YES-OR-NO.
TELL-ACTION25 was not grounded in INDICATE-YES-OR-NO because:
 Can't concrete UTTERANCE22 to UTTERANCE-OF-INDICATE-YES-OR-NO because the ranges differ.
Determining whether TELL-ACTION25 is a new plan for the existing goal of KNOWING-WHETHER6
TELL-ACTION25 is not a new plan for the existing goal of KNOWING-WHETHER6 because:
 Attempt to concrete incompatibly from TELL-ACTION25 to KNOWING-WHETHER6.
TELL-ACTION25 was not grounded in the preceding dialogue.

Found PLANFOR3 as an explanation for TELL-ACTION25.
Synopsis:
 Goal: hearer know whether X;
 Step 1: Tell hearer that X.
Inferred goal is KNOWING-WHETHER7.

Determining whether TELL-ACTION26 is grounded in the preceding dialogue.
Trying to ground TELL-ACTION26 in INDICATE-YES-OR-NO.
 Concreting TELL-ACTION26 to be INDICATE-YES-OR-NO.
TELL-ACTION26 was not grounded in INDICATE-YES-OR-NO because:
 Can't concrete UTTERANCE23 to UTTERANCE-OF-INDICATE-YES-OR-NO because the ranges differ.
Determining whether TELL-ACTION26 is a new plan for the existing goal of KNOWING-WHETHER6
TELL-ACTION26 is not a new plan for the existing goal of KNOWING-WHETHER6 because:
 Attempt to concrete incompatibly from TELL-ACTION26 to KNOWING-WHETHER6.
TELL-ACTION26 was not grounded in the preceding dialogue.

Found PLANFOR3 as an explanation for TELL-ACTION26.
Synopsis:
 Goal: hearer know whether X;
 Step 1: Tell hearer that X.
Inferred goal is KNOWING-WHETHER8.

Attempting to resolve any ambiguities in KNOWING-WHETHER8
 Retrieving potential conflicts with KNOWING-WHETHER8 and its plan.
 No conflict found.

Attempting to resolve any ambiguities in KNOWING-WHETHER7
 Retrieving potential conflicts with KNOWING-WHETHER7 and its plan.
 No conflict found.

> PAGAN *now tries to determine whether either of the inferred goals matches an expectation, or constitutes a new plan for an existing goal. The goal of* UC *knowing whether the file is in the user's directory is matched, so the interpretation that binds the word 'it' to the file is selected. The other interpretation is rejected.*

Determining whether KNOWING-WHETHER7 is grounded in the preceding dialogue.
Trying to ground KNOWING-WHETHER7 in INDICATE-YES-OR-NO.
KNOWING-WHETHER7 was not grounded in INDICATE-YES-OR-NO because:
 Attempt to concrete incompatibly from KNOWING-WHETHER7 to INDICATE-YES-OR-NO.
Determining whether KNOWING-WHETHER7 is a new plan for the existing goal of KNOWING-WHETHER6
KNOWING-WHETHER7 is a new plan for the existing goal of KNOWING-WHETHER6
Existing plan PLAN58 has been rejected; new plan is PLAN62.
Selecting KNOWING-WHETHER7 as the correct explanation of TELL-ACTION25
Selecting TELL-ACTION25 as the correct explanation of (IT IS IN /TMP)

Rejecting TELL-ACTION26 as an explanation of (IT IS IN /TMP).
Rejecting KNOWING-WHETHER8 as an explanation of TELL-ACTION26.

7.4.3. Delaying Ambiguity Resolution

The delay of ambiguity resolution is a natural outcome of the goal analysis algorithm presented here. If none of the above techniques for disambiguation is successful, each of the potential explanations is extended individually. Thus, ambiguity resolution is automatically delayed until some further explanation is found. If the chaining algorithm described in Chapter 8 determines that no further processing should be done on an utterance that exhibits an apparent ambiguity, then the ambiguity is left as real ambiguity.

Processing Example

This trace shows PAGAN's processing of the sequence:

User: I want more disk space.
User: Will mv delete a file?

PAGAN detects the topic ambiguity in the user's second utterance. By disregarding this ambiguity for several iterations of the goal analysis algorithm, it eventually resolves the ambiguity by relating it to the user's goal, stated in the first utterance, of having more disk space.

User: I want more disk space.

User: Will mv delete a file?

> *The semantic interpreter detects topic ambiguity in this utterance, and builds two competing interpretations. In the first interpretation, the* mv *command is the topic (that is, it is the argument of the* WHAT-IS*); in the second, file deletion is the topic.*

The parser produced the following interpretations:

```
Interpretation 1 (TELL-ACTION28):
(HAS-GOAL54 (DOMAIN USER)
 (RANGE
  (TELL-ACTION28 (DOMINATED BY ASK-VERIFICATION-QUESTION)
   (ACTOR35 = USER) (HEARER25 = UNIX-CONSULTANT) (SPEAKER24 = USER)
   (UTTERANCE26 =
    (VERIFICATION-QUESTION8 (DOMINATED BY VERIFICATION-QUESTION)
     (WHAT-IS19 =
```

(MV
(ACTOR34* =
(DELETE-ACTION13 (DOMINATED BY FILE-DELETE-ACTION)
(CAUSES-OF-DELETE-ACTION16 =
(DELETION19 (DOMINATED BY FILE-DELETION)
(STATE-CHANGE-OBJECT14 = (FILE17 (DOMINATED BY FILE))))))))))))))

Interpretation 2 (TELL-ACTION29):
(HAS-GOAL55 (DOMAIN USER)
(RANGE
(TELL-ACTION29 (DOMINATED BY ASK-VERIFICATION-QUESTION)
(ACTOR37 = USER) (HEARER26 = UNIX-CONSULTANT) (SPEAKER25 = USER)
(UTTERANCE27 =
(VERIFICATION-QUESTION9 (DOMINATED BY VERIFICATION-QUESTION)
(WHAT-IS20 =
(DELETE-ACTION14 (DOMINATED BY FILE-DELETE-ACTION) (ACTOR36 = MV)
(CAUSES-OF-DELETE-ACTION17 =
(DELETION20 (DOMINATED BY FILE-DELETION)
(STATE-CHANGE-OBJECT15 = (FILE18 (DOMINATED BY FILE))))))))))))

Unable to resolve the ambiguity immediately, PAGAN *continues its processing.*

Attempting to resolve any ambiguities in TELL-ACTION29
Retrieving potential conflicts with TELL-ACTION29 and its plan.
No conflict found.

Attempting to resolve any ambiguities in TELL-ACTION28
Retrieving potential conflicts with TELL-ACTION28 and its plan.
No conflict found.

The NO-STEP *portion of the representation of the plan and goal structure of the user's first utterance is simply a dummy plan step that never matches any real event.*

Determining whether TELL-ACTION28 is grounded in the preceding dialogue.
Trying to ground TELL-ACTION28 in NO-STEP.
TELL-ACTION28 was not grounded in NO-STEP because:
Attempt to concrete incompatibly from TELL-ACTION28 to NO-STEP.
Determining whether TELL-ACTION28 is a new plan for the existing goal of
INCREASED-DISK-SPACE2
TELL-ACTION28 is not a new plan for the existing goal of INCREASED-DISK-SPACE2
because:
Attempt to concrete incompatibly from TELL-ACTION28 to INCREASED-DISK-SPACE2.
TELL-ACTION28 was not grounded in the preceding dialogue.

Found PLANFOR2 as an explanation for TELL-ACTION28.
Synopsis:
Goal: Know whether f(X) = Y;
Step 1: Ask whether f(X) = Y;
Step 2: Hearer says yes or no.
Inferred goal is KNOWING-WHETHER10.

Determining whether TELL-ACTION29 is grounded in the preceding dialogue.
Trying to ground TELL-ACTION29 in NO-STEP.
TELL-ACTION29 was not grounded in NO-STEP because:
Attempt to concrete incompatibly from TELL-ACTION29 to NO-STEP.
Determining whether TELL-ACTION29 is a new plan for the existing goal of
INCREASED-DISK-SPACE2
TELL-ACTION29 is not a new plan for the existing goal of INCREASED-DISK-SPACE2
because:
Attempt to concrete incompatibly from TELL-ACTION29 to INCREASED-DISK-SPACE2.
TELL-ACTION29 was not grounded in the preceding dialogue.

Found PLANFOR2 as an explanation for TELL-ACTION29.
Synopsis:
Goal: Know whether f(X) = Y;
Step 1: Ask whether f(X) = Y;
Step 2: Hearer says yes or no.
Inferred goal is KNOWING-WHETHER11.

Again, PAGAN *tries to resolve the ambiguity, but is unsuccessful. Processing is continued.*

Attempting to resolve any ambiguities in KNOWING-WHETHER11
Retrieving potential conflicts with KNOWING-WHETHER11 and its plan.
No conflict found.

Attempting to resolve any ambiguities in KNOWING-WHETHER10
Retrieving potential conflicts with KNOWING-WHETHER10 and its plan.
No conflict found.

Determining whether KNOWING-WHETHER10 is grounded in the preceding dialogue.
Trying to ground KNOWING-WHETHER10 in NO-STEP.
KNOWING-WHETHER10 was not grounded in NO-STEP because:
Attempt to concrete incompatibly from KNOWING-WHETHER10 to NO-STEP.
Determining whether KNOWING-WHETHER10 is a new plan for the existing goal of
INCREASED-DISK-SPACE2
KNOWING-WHETHER10 is not a new plan for the existing goal of INCREASED-DISK-SPACE2
because:
Attempt to concrete incompatibly from KNOWING-WHETHER10 to
INCREASED-DISK-SPACE2. KNOWING-WHETHER10 was not grounded in the preceding dialogue.

Found PLANFOR1 as an explanation for KNOWING-WHETHER10.
Synopsis:
Goal: Know f(X);
Step 1: Know that f(X) = Y
Inferred goal is KNOWING-THAT9.

Determining whether KNOWING-WHETHER11 is grounded in the preceding dialogue.
Trying to ground KNOWING-WHETHER11 in NO-STEP.
KNOWING-WHETHER11 was not grounded in NO-STEP because:
Attempt to concrete incompatibly from KNOWING-WHETHER11 to NO-STEP.
Determining whether KNOWING-WHETHER11 is a new plan for the existing goal of INCREASED-DISK-SPACE2
KNOWING-WHETHER11 is not a new plan for the existing goal of INCREASED-DISK-SPACE2 because:
Attempt to concrete incompatibly from KNOWING-WHETHER11 to INCREASED-DISK-SPACE2. KNOWING-WHETHER11 was not grounded in the preceding dialogue.

Found PLANFOR1 as an explanation for KNOWING-WHETHER11.
Synopsis:
Goal: Know f(X);
Step 1: Know that f(X) = Y
Inferred goal is KNOWING-THAT10.

The ambiguity still cannot be resolved.

Attempting to resolve any ambiguities in KNOWING-THAT10
Retrieving potential conflicts with KNOWING-THAT10 and its plan.
No conflict found.

Attempting to resolve any ambiguities in KNOWING-THAT9
Retrieving potential conflicts with KNOWING-THAT9 and its plan.
No conflict found.

Determining whether KNOWING-THAT9 is grounded in the preceding dialogue.
Trying to ground KNOWING-THAT9 in NO-STEP.
KNOWING-THAT9 was not grounded in NO-STEP because:
Attempt to concrete incompatibly from KNOWING-THAT9 to NO-STEP.
Determining whether KNOWING-THAT9 is a new plan for the existing goal of INCREASED-DISK-SPACE2
KNOWING-THAT9 is not a new plan for the existing goal of INCREASED-DISK-SPACE2 because:
Attempt to concrete incompatibly from KNOWING-THAT9 to INCREASED-DISK-SPACE2.
KNOWING-THAT9 was not grounded in the preceding dialogue.

Here, PAGAN *is hypothesizing that the user may want to know how to use* mv *so as to do whatever it is that* mv *does. Since* PAGAN *doesn't know exactly which effect of* mv *is intended, it simply produces the generic* EVENT14 *to represent this effect.*

Found PLANFOR8 as an explanation for KNOWING-THAT9.

Synopsis:
Goal: Achieve X;
Step 1: Know how to X;
Step 2: Do X
Inferred goal is EVENT14.

Determining whether KNOWING-THAT10 is grounded in the preceding dialogue.
Trying to ground KNOWING-THAT10 in NO-STEP.
KNOWING-THAT10 was not grounded in NO-STEP because:
Attempt to concrete incompatibly from KNOWING-THAT10 to NO-STEP.
Determining whether KNOWING-THAT10 is a new plan for the existing goal of INCREASED-DISK-SPACE2
KNOWING-THAT10 is not a new plan for the existing goal of INCREASED-DISK-SPACE2 because:
Attempt to concrete incompatibly from KNOWING-THAT10 to INCREASED-DISK-SPACE2.
KNOWING-THAT10 was not grounded in the preceding dialogue.

The goal of knowing how to delete a file is explained here by the goal of actually deleting a file.

Found PLANFOR8 as an explanation for KNOWING-THAT10.
Synopsis:
Goal: Achieve X;
Step 1: Know how to X;
Step 2: Do X
Inferred goal is DELETION22.

The ambiguity persists.

Attempting to resolve any ambiguities in DELETION22
Retrieving potential conflicts with DELETION22 and its plan.
No conflict found.

Attempting to resolve any ambiguities in EVENT14
Retrieving potential conflicts with EVENT14 and its plan.
No conflict found.

Determining whether EVENT14 is grounded in the preceding dialogue.
Trying to ground EVENT14 in NO-STEP.
EVENT14 was not grounded in NO-STEP because:
Attempt to concrete incompatibly from EVENT14 to NO-STEP.
Determining whether EVENT14 is a new plan for the existing goal of INCREASED-DISK-SPACE2
EVENT14 is not a new plan for the existing goal of INCREASED-DISK-SPACE2 because:
Attempt to concrete incompatibly from EVENT14 to INCREASED-DISK-SPACE2.
EVENT14 was not grounded in the preceding dialogue.

Since EVENT14 *is simply a dummy effect of the* mv *command,* PAGAN *finds no explanation for it.*

No explanation found for EVENT14.

Determining whether DELETION22 is grounded in the preceding dialogue.
Trying to ground DELETION22 in NO-STEP.
DELETION22 was not grounded in NO-STEP because:
Attempt to concrete incompatibly from DELETION22 to NO-STEP.
Determining whether DELETION22 is a new plan for the existing goal of INCREASED-DISK-SPACE2
DELETION22 is not a new plan for the existing goal of INCREASED-DISK-SPACE2 because:
Attempt to concrete incompatibly from DELETION22 to INCREASED-DISK-SPACE2.
DELETION22 was not grounded in the preceding dialogue.

File deletion is a plan for gaining increased disk space. For this example, increased disk space is the only effect of file deletion that has been included in the knowledge base.

Found PLANFOR14 as an explanation for DELETION22.
Synopsis:
Goal: increased disk space;
Step 1: delete a file
Inferred goal is INCREASED-DISK-SPACE3.

Attempting to resolve any ambiguities in INCREASED-DISK-SPACE3
Retrieving potential conflicts with INCREASED-DISK-SPACE3 and its plan.
No conflict found.

Finally, the 'file' interpretation is connected to the plan and goal structure produced during the processing of the user's first statement. This interpretation is selected, while the 'directory' interpretation is rejected.

Determining whether INCREASED-DISK-SPACE3 is grounded in the preceding dialogue.
Trying to ground INCREASED-DISK-SPACE3 in NO-STEP.
INCREASED-DISK-SPACE3 was not grounded in NO-STEP because:
Attempt to concrete incompatibly from INCREASED-DISK-SPACE3 to NO-STEP.
Determining whether INCREASED-DISK-SPACE3 is a new plan for the existing goal of INCREASED-DISK-SPACE2
INCREASED-DISK-SPACE3 is a new plan for the existing goal of INCREASED-DISK-SPACE2
Existing plan PLAN64 has been rejected; new plan is PLAN74.
Selecting INCREASED-DISK-SPACE3 as the correct explanation of DELETION22
Selecting DELETION22 as the correct explanation of KNOWING-THAT10
Selecting KNOWING-THAT10 as the correct explanation of KNOWING-WHETHER11
Selecting KNOWING-WHETHER11 as the correct explanation of TELL-ACTION29
Selecting TELL-ACTION29 as the correct explanation of (WILL MV DELETE A FILE)
Rejecting TELL-ACTION28 as an explanation of (WILL MV DELETE A FILE).
Rejecting KNOWING-WHETHER10 as an explanation of TELL-ACTION28.
Rejecting KNOWING-THAT9 as an explanation of KNOWING-WHETHER10.
Rejecting EVENT14 as an explanation of KNOWING-THAT9.

7.5. Summary

There are three ways to handle ambiguity, regardless of the source of that ambiguity. First, one of the interpretations can be selected as the correct one. The most important method for selecting the correct interpretation, and one that has been widely used, is to match that interpretation against an expected event. Secondly, interpretations can be rejected as incorrect until only the correct interpretation remains. Rejection is based on detecting a conflict between the interpretation and some other knowledge held by the system. This chapter provided a framework for examining such 'knowledge conflicts.' Six types of conflict that might hold between an interpretation of an utterance and the user model were described. Finally, delaying the resolution of an ambiguity to some later time is an acceptable way to handle ambiguity.

Chapter 8

Extending an Explanation

The third main step of the goal analysis algorithm is to determine whether the analysis of the utterance should be continued. To continue the analysis of a particular utterance is to extend a chain of inferences about that utterance by inferring an additional planfor to explain the most recently inferred goal. I call such a chain of inferences an *explanation chain.*

My purpose in this chapter is threefold. First, I discuss previous approaches to the problem of determining when to continue analysis, and show how certain of these approaches are flawed. Secondly, I introduce an algorithm for determining when goal analysis should continue. This algorithm is based on determining whether the explanation is complete (as described in Chapter 3). Finally, I demonstrate how an outside process can explicitly control chaining on an explanation.

8.1. Previous Approaches

Consider the utterance:

> **User:** Can you tell me how to edit a file?

One possible explanation chain of this question contains the following goals:

1. The user wants to request that you tell her how to delete a file
2. The user wants to know how to edit a file
3. The user wants to edit a file
4. The user wants to insert the text of a paper into a file
5. The user wants to print a paper
6. The user wants to publish a paper
7. The user wants to get tenure

Each of these goals represents the result of one iteration of the plan recognition algorithm. The first goal directly explains the utterance. Each of the subsequent goals provides an explanation for the preceding goal, and therefore provides part of an explanation for the utterance.

A system that is designed to be general enough to handle a wide range of inputs will represent the knowledge needed to make some or all of these inferences separately. However, not all of these explanations will be applicable to every system that must handle the utterance in question. Consequently, PAGAN must determine which of these inferences it should make, and which it should avoid making. Since each inference is built on the preceding inference, the decision that must be made is when the chaining process should continue, and when it should stop.

Some systems make every possible inference. This is an acceptable approach because their domains are highly constrained. In a less constrained environment in which the system must deal with a wide range of human affairs, this approach is not tenable. This is true even if the particular topic of discussion is quite limited, because the system will still need access to a wide range of world knowledge. Therefore, some method for determining when to cut off the inference process is needed.

The following rules have been proposed for deciding when to halt chaining:

Rule 1. Halt when the plan inferred serves a known goal. [Mann et al. 1977]
Rule 2. Halt when a thematic explanation is found [Wilensky 1978]
Rule 3. Halt when the goal is uninteresting [Schank 1979]
Rule 4. Halt when a meta-plan to a known plan is found [Litman 1985]
Rule 5. Halt when a domain goal is encountered. [Litman 1985]
Rule 6. Halt when an ambiguity is encountered. [Sidner 1985]

Rule 1 says that chaining should stop when an inferred explanation matches some expected goal. Rule 2 introduces themes to serve as high-level expectations, thereby allowing Rule 1 to be applicable when there has been no preceding dialogue. Rule 3 proposes that the decision about when to chain should be based not only on the content of the utterance, but also on the hearer's attitudes towards the utterance and the inferred goals. Rule 4 expands the notion of how an utterance may connect to previous dialogue by viewing it as a meta-plan to some existing dialogue plan. Rule 5 supports the position that chaining should stop as soon as some goal in the domain of discourse is inferred. Finally, Rule 6 says that chaining should stop when two or more explanations of the same event are found that cannot be disambiguated immediately.

Each of rules 1-4 will be incorporated into the goal analysis algorithm in the following sections. Rules 5 and 6 concern the termination of chaining when the explanation cannot be grounded in knowledge of the preceding dialogue. Each of these rules is problematic. Rule 5 is problematic because the first domain goal encountered may not be the important one. Consider the question:

User: Can you tell me how I can compact my directory?

The first domain goal that is encountered in the analysis of this utterance is the goal of compacting a directory. If analysis is halted once this goal is inferred, there is no way that the system can suggest for example that the user request a larger disk quota from the system manager. This is because to do so would require that the system know of the user's goal of gaining more disk space. If chaining is stopped once the goal of compacting the directory has been inferred, this motivating domain goal will never be discovered.

Rule 6 is also problematic. Sidner's reason for discontinuing processing when an ambiguity is detected is that a participant in a dialogue cannot expect a hearer to react to an utterance based on "the assumption that somehow the hearer will guess at the information that is needed for the hearer to make an intended response" [p. 4]. This is true; however, it does not follow that processing should be halted when an ambiguity is encountered. There are two reasons that it may be important to continue analysis after an ambiguity has been discovered. First, such subsequent processing may itself prove important for the analysis of the utterance. The best example of this is when further processing allows the ambiguity to be resolved. This phenomenon was discussed in Chapter 7.

The second reason that it may be important to continue processing after an ambiguity is detected is that the speaker may not be aware that the ambiguity is present. A speaker never has a perfect model of the hearer. Consequently, an utterance that is believed by its speaker to be unambiguous may be perceived by a hearer as ambiguous. In such cases, it may be important for the hearer to fully develop the competing interpretations, even if the ambiguity cannot be resolved. Doing so gives the components of the system outside of the goal analyzer the greatest latitude in deciding which goals to address.

The approach to chaining implicit in these rules bases the decision to chain on each competing explanation chain individually. That is, when a goal is inferred, the decision about whether that goal must itself be explained does not depend on what other potential explanation chains are extant. In contrast to this approach is the resource-based method, exemplified by Allen [1979]. In this method, decisions are made not about whether an explanation for a goal is needed, but rather about where processing power should be directed. Allen's system for example had a set of heuristics for rating the strength of an explanation. Processing was done on the explanation with the highest rating. Thus the decision about whether to chain on a particular explanation was based not only on the attributes of that explanation, but also on attributes of all other explanations under consideration.

There are two main problems with this resource-based approach to chaining. First, it fails when a single utterance is made in service of multiple goals. In such cases, this approach will find only one chain of goals that motivated the utterance. Secondly, this approach assumes that the processor is a scarce resource. With the development of parallel architectures and languages, it is becoming increasingly feasible to carry forward multiple lines of reasoning. A procedure for determining when to chain should be able to

take advantage of such parallelism.

The rules described above represent pieces of the solution to the chaining problem. What has been lacking in the literature is not so much heuristics to apply to the chaining problem as a framework that ties together the available heuristics. In the following section I propose such a framework. I then show how the valid heuristics itemized above (rules 1-4), as well as some new heuristics, can be placed into this framework to form a complete approach to the chaining problem.

8.2. Deciding When to Chain

The framework for deciding when to chain that I propose is based on two assessments:

1. An assessment of the completeness of the explanation chain
2. An assessment of the practicability of chaining

The decision about whether to chain is dominated (but not dictated) by whether PAGAN believes the explanation chain to be *complete*. I refer here to the modified principle of depth completeness described in Chapter 3. If the explanation chain is complete, there is no reason to continue trying to explain the utterance. If the explanation chain is not complete, then under ideal circumstances processing should continue until it is complete. Thus, the first step in deciding whether to chain is to determine whether the explanation chain in question is complete.

Of course, it is not always possible to determine whether an explanation chain is complete. Thus, the function that examines an explanation chain to see whether it is complete must indicate that one of three conditions holds:

1. The explanation chain is complete
2. The explanation chain is incomplete
3. It cannot be determined whether the explanation chain is complete

Once the determination has been made as to whether an explanation chain is complete, that knowledge must be used to decide whether to continue processing. Ideally, processing would stop if the explanation chain were complete, and would continue otherwise. This would result in the explanation meeting the completeness criterion described in Chapter 3. However, the difficulty of determining whether an explanation chain is complete, together with resource limitations, prevent this strategy from being a practical one. Consequently, if it can be determined that the explanation chain is certainly complete, or that the explanation chain is certainly incomplete, the decision about whether to chain is based on this determination. However, if it cannot be determined whether the explanation chain is complete, PAGAN must decide whether to chain by assessing the benefits of chaining and comparing these benefits against the cost of chaining. If the benefits outweigh the cost, chaining continues. However, if the converse holds, chaining halts. This limits computational expense, at the cost of producing an explanation that is

not as good according to the criteria described in Chapter 3.

The completeness of an explanation chain is largely based on the characteristics of that explanation chain. In contrast, the cost/benefit analysis of chaining is primarily based on the status of PAGAN's processing. The cost of chaining is determined by estimating the amount of work that will be required to find all reasonable explanations for the goal in question. The benefits of chaining are determined by assessing PAGAN's *curiosity* about the goal. Both of these estimates rely not only on knowledge of the goal to be explained, but also on knowledge of the system's goals and capabilities.

When PAGAN cannot determine whether an explanation chain is complete or incomplete, it estimates the costs and the benefits of chaining. These estimates are used in the following way. If the strength of one exceeds the strength of the other by a threshold, the stronger one is given precedence. For example, if the curiosity level is high, and the cost of continuing the analysis appears to be low, chaining continues. If on the other hand the two are of roughly equal strength, then analysis is discontinued.

In the following subsections, I first show how an explanation chain may be deemed to be complete or incomplete. I then discuss how a cost/benefits analysis of chaining may be performed. In the section following this one, I describe how an outside process can request the continuation of an analysis that this algorithm has determined should be discontinued.

8.2.1. Determining that an Explanation Chain is Complete

There are two conditions that indicate to PAGAN that an explanation chain is complete:

1. The explanation chain has previously been determined to be complete
2. The explanation chain meets established criteria for completeness

First, the explanation chain may already have been determined to be complete. This is the case when an explanation chain matches an expectation. Expectation matching was described in Chapter 5 as a way to connect an utterance to a representation of the preceding dialogue, thereby grounding the utterance in the dialogue. Since PAGAN already has an adequate explanation for any action that it expects, no further work is required to satisfy its needs in this situation. Note that if metaplans are viewed as unexpected steps of existing plans, they can be handled during expectation matching.

The second way that PAGAN may decide that an explanation chain is complete is by applying established completeness criteria. Such criteria are derived from knowledge of the system's own domain of expertise. This use of knowledge of the domain of expertise, that is, knowledge of the system's contract with the user, stems directly from the principle of applicability discussed in Chapter 3. An explanation chain is declared to be complete when the most recently inferred goal is no longer applicable to the purposes of the system. Consider for example the following question:

User: How can I read a phone number that's in Beth's directory?

An explanation of this utterance might contain the goal of making a phone call. Such an explanation chain is complete for UC (whether or not it meets the other criteria for good explanations) because the goal of making a phone call is not within the UC domain.

8.2.2. Determining that an Explanation Chain is Incomplete

In addition to determining that an explanation chain is complete, it may also be possible to determine that it is incomplete. There are two approaches to judging the incompleteness of an explanation chain:

1. Examination of the character of the explanation itself
2. Examination of expectations derived from the preceding dialogue

Some goals seem to fall naturally under the rubric of incomplete explanations. Such a goal is called an *instrumental goal.* An instrumental goal is one that arises only to serve another goal [Schank and Abelson 1977]. Schank and Abelson's concept of an instrumental goal is an absolute one; a particular goal is either an instrumental goal or it isn't. Thus, getting a loan is always instrumental to using the money. In contrast to this position, I claim that what is instrumental is relative to the task at hand. Editing a file is instrumental to writing a dissertation, but at the level at which UC works, it may be a top-level goal.

The second way that an explanation chain may be judged to be incomplete is based on knowledge of the preceding dialogue. When there is a strong expectation of a particular action, and the explanation chain built so far neither matches that expectation nor is grounded in the previous dialogue in any way, incompleteness is indicated. This provides a strong impetus to treat the next utterance as a means of achieving the expected action. Consider again this exchange:

UC: Is the file in your directory?
User: The file is in /tmp.

Here there is a strong expectation that the user's utterance will somehow relate to UC's question. Consequently, chaining should continue on this utterance until it is determined that the user's utterance constitutes a new plan for UC's goal (as described in Chapter 5).

8.2.3. Determining the Cost of Chaining -- Difficulty of Analysis

It is sometimes expedient to save processing time at the expense of producing a poorer explanation of an utterance. The principle of *difficulty of analysis* holds that analysis of an utterance should be terminated if the cost of continuing the analysis becomes too high. Consider for example the question:

User: How can I edit a file?

In the absence of other knowledge, the goal of editing a file is a difficult one to explain. There are many reasons that one might want to edit a file. It would be prohibitively expensive to carry out an analysis of each of these explanation chains. The principle of difficulty of analysis provides an impetus for the goal analyzer to terminate processing in such situations.

8.2.4. Determining the Benefits of Chaining -- Curiosity

The benefits of chaining on an explanation chain that is clearly incomplete are obvious when one's goal is to produce complete explanations. Chaining is always done on such explanation chains. When the explanation chain in question is not clearly incomplete though, PAGAN must base the chaining decision not on the absolute characteristics of the explanation, but rather with how those characteristics relate to the goals of the system. The *principle of curiosity* defines this relationship. This principle directs PAGAN to continue analysis on goals about which it is curious. I define *interestingness* as that quality that causes PAGAN to be curious about a goal [cf. Schank 1979].

A goal is deemed to be interesting when it is in an interesting goal relationship with another goal. Wilensky [1983] identifies four kinds of goal relationships: goal overlap, goal concord, goal conflict, and goal competition. Any of these types of goal relationships can give rise to curiosity based on interestingness. I will focus here on goal competition (which is a negative relationship between the goals of two different agents).

Goal competition is defined by Wilensky as a situation in which "the fulfillment of one [agent's] goals precludes the fulfillment of another's" [1983, p. 58]. When a speaker's goal comes into competition with a goal held by the system, PAGAN should be curious about that goal. For example, if the user asks:

User: How do I change the 'cat' program?

PAGAN may infer that the user wants to change the cat[1] program. This goal competes with UC's goal of preserving system programs. Because of this competition, PAGAN will be curious about why the user would want to change the cat program. It will continue its analysis, possibly arriving at the motivating goal of having a program that is similar in effect to the cat program. This allows UC to propose the 'copy, then change' plan:

UC: First copy the source code with cp, then edit and compile the copy.

The importance of curiosity can be seen by examining the processing of the question:

[1] Cat is a system program in UNIX that displays the contents of a file.

User: How do I change a.out?[2]

In this example, there may be many ways to change a.out. To select the correct one, the planner needs to know what goal motivates the speaker to want to change a.out. This motivating goal is most likely to be the alteration of the behavior of the program. However, if the planner isn't aware of that goal, it may well choose a different plan, such as:

UC: Use gnumacs.

Using an editor on executable code, while it could theoretically be used to effect a desired programmatic change, is in most cases a ridiculous plan for doing so. An appropriate response would be:

UC: Edit the source code, then recompile it.

The point here is that inferring a higher-level goal may allow the concretion of a lower-level goal. In this example, inferring the goal of changing the program's behavior allows a concretion to be made from 'changing a.out' to 'changing the behavior of a.out.' Since the planner may have more specific plans for more specific goals, the performance of such a concretion may reveal a better plan.

8.3. External Control of Chaining

The amount of work that **PAGAN** does on an explanation of an utterance can also be controlled by outside processes. This is achieved by allowing other processes to re-invoke **PAGAN** on an utterance, indicating that more of an explanation of the utterance is needed. Consider the question:

User: How do I edit one of Barney's files?

PAGAN's processing of this question is initially simple. It starts by inferring that the user wants to edit a file owned by Barney. At this point, **PAGAN** determines that the goal it has inferred is neither obviously instrumental, nor intrinsically interesting. Therefore, **PAGAN** halts its processing of the utterance. **UC** then calls **KIP** [Luria 1988] to plan for the goal of editing Barney's file. **KIP** tries to do so, but fails. The reason is that, in UNIX, typically only the owner of a file may edit that file. Instead of exerting great energy to find *some* convoluted plan to achieve this goal, **KIP** simply re-invokes **PAGAN**, indicating that it would prefer to plan for some other goal. It is easy for **PAGAN** to find a parent goal, namely that the user wants a file with contents similar to those of Barney's file.[3] **KIP** now has a goal with a well-known plan, and so produces the 'copy file, then edit the copy' plan.

---

[2] In UNIX, a.out is a common name for an executable program.

[3] This is not a parent goal in every context, but making that determination is a separate issue.

This complicated flow of control is advantageous for two reasons. First, when the initial goal cannot easily be planned for, the savings in planning time can be considerable. In this example, if KIP worked hard, it might be able to construct a plan for the goal initially suggested by PAGAN (for instance, 'boot the machine that is acting as the file server single user, login as root, then edit the file'). Or, KIP might continue planning *ad infinitum* for a goal that it cannot solve. If PAGAN can infer a parent goal for which it is easy to construct a plan, as in this example, such wasted planning can be avoided.

The second benefit of this control structure is that in cases where the initial goal can be planned for easily, goal analysis time is reduced. This savings can be significant; continuation of goal analysis, once a reasonable goal has been inferred, is often expensive.

Processing Example

This trace shows how PAGAN can be re-invoked on an analysis that it had initially terminated. The utterance being processed is:

User: How do I edit Barney's file?

The analysis proceeds until the goal of editing the file is inferred, at which point PAGAN terminates its processing. PAGAN is then re-invoked to continue its analysis. It goes on to infer the goal of obtaining a changed version of the contents of the file.

User: How do I edit Barney's file?

Interpretation produced by the parser: TELL-ACTION34

Chaining is continued on this utterance until the domain goal of editing Barney's file has been inferred.

Determining whether to chain on TELL-ACTION34.
  Determining whether TELL-ACTION34 is a complete explanation of
  (HOW DO I EDIT BARNEY'S FILE).
    Determining whether TELL-ACTION34 is outside of UC's domain of expertise.
    TELL-ACTION34 is not outside of UC's domain of expertise.
  The explanation is not necessarily complete.
  Determining whether TELL-ACTION34 is an incomplete explanation of
  (HOW DO I EDIT BARNEY'S FILE).
    Determining whether TELL-ACTION34 is an instrumental goal.
    TELL-ACTION34 is an instrumental goal.
  The explanation is incomplete.
Chaining should be done on TELL-ACTION34.

Determining whether TELL-ACTION34 is grounded in the preceding dialogue.
TELL-ACTION34 was not grounded in the preceding dialogue.

Found PLANFOR4 as an explanation for TELL-ACTION34.
Synopsis:
Goal: know something;
Step 1: ask;
Step 2: be told.
Inferred goal is KNOWING-THAT13.

Determining whether to chain on KNOWING-THAT13.
Determining whether KNOWING-THAT13 is a complete explanation of TELL-ACTION34.
Determining whether KNOWING-THAT13 is outside of UC's domain of expertise.
KNOWING-THAT13 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether KNOWING-THAT13 is an incomplete explanation of TELL-ACTION34.
Determining whether KNOWING-THAT13 is an instrumental goal.
KNOWING-THAT13 is an instrumental goal.
The explanation is incomplete.
Chaining should be done on KNOWING-THAT13.

Determining whether KNOWING-THAT13 is grounded in the preceding dialogue.
KNOWING-THAT13 was not grounded in the preceding dialogue.

Found PLANFOR8 as an explanation for KNOWING-THAT13.
Synopsis:
Goal: Achieve X;
Step 1: Know how to X;
Step 2: Do X
Inferred goal is EDITING3.

PAGAN *now determines whether it should try to figure out why the user might want to edit Barney's file (as represented by* EDITING3*). It decides against continuing its processing, because the estimated costs of continuing outweigh its interest in continuing. Consequently, processing is halted.*

Determining whether to chain on EDITING3.
Determining whether EDITING3 is a complete explanation of KNOWING-THAT13.
Determining whether EDITING3 is outside of UC's domain of expertise.
EDITING3 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether EDITING3 is an incomplete explanation of KNOWING-THAT13.
Determining whether EDITING3 is an instrumental goal.
EDITING3 is not an instrumental goal.
Determining whether there is a strong expectation for a particular action.
There is no strong expectation for a particular action.
The explanation is not necessarily incomplete.
Determining the level of curiosity about EDITING3.

Determining the level of interest in EDITING3.
Interest level = 0.
Determining the level to which the goal EDITING3 conflicts with system goals.
Conflict level = 0.
Curiosity level = 0.
Determining estimated cost of chaining on EDITING3.
Estimated cost = 1.
Chaining should not be done on EDITING3.

PAGAN *is now re-invoked by an outside process. It continues its processing of the utterance where it left off.*

Re-invoking PAGAN to continue its analysis.

Determining whether EDITING3 is grounded in the preceding dialogue.
EDITING3 was not grounded in the preceding dialogue.

PAGAN *finds the goal of obtaining access to the altered contents of the file as an explanation of* EDITING3. *This goal is easier to plan for than the goal of editing someone else's file; one can simply copy the file, then edit the copy.*

Found PLANFOR10 as an explanation for EDITING3.
Synopsis:
Goal: have file with changed contents;
Step 1: edit file
Inferred goal is OBTAIN-CHANGED-CONTENTS0.

Determining whether to chain on OBTAIN-CHANGED-CONTENTS0.
Determining whether OBTAIN-CHANGED-CONTENTS0 is a complete explanation of EDITING3.
Determining whether OBTAIN-CHANGED-CONTENTS0 is outside of UC's domain of expertise.
OBTAIN-CHANGED-CONTENTS0 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether OBTAIN-CHANGED-CONTENTS0 is an incomplete explanation of EDITING3.
Determining whether OBTAIN-CHANGED-CONTENTS0 is an instrumental goal.
OBTAIN-CHANGED-CONTENTS0 is not an instrumental goal.
Determining whether there is a strong expectation for a particular action.
There is no strong expectation for a particular action.
The explanation is not necessarily incomplete.
Determining the level of curiosity about OBTAIN-CHANGED-CONTENTS0.
Determining the level of interest in OBTAIN-CHANGED-CONTENTS0.
Interest level = 0.
Determining the level to which the goal OBTAIN-CHANGED-CONTENTS0 conflicts with system goals.
Conflict level = 0.
Curiosity level = 0.

Determining estimated cost of chaining on OBTAIN-CHANGED-CONTENTS0.
Estimated cost = 1.
Chaining should not be done on OBTAIN-CHANGED-CONTENTS0.

8.4. Implementation

In the following sections, I discuss some of the implementation details of the concepts discussed in this chapter. First, I cover how PAGAN can determine whether a goal is instrumental, and whether it is applicable. Next, I discuss the identification of a strong expectation for a particular action. Finally, I discuss the two components of determining the practicability of chaining: difficulty of analysis and curiosity. The implementation of expectation matching was discussed in Chapter 5.

8.4.1. Determining Instrumentality and Applicability

The most general approach to determining whether a goal is within the system's domain of expertise would take into consideration not only the goal itself and the purposes of the system, but also knowledge of the user and of the preceding dialogue. In practice though, it is more efficient, albeit less flexible, simply to mark explicitly each type of goal that is within the system's domain of expertise. This is the technique that the implementation of PAGAN uses to determine which goals are in UC's domain of discourse. While this technique cannot change its evaluations as the topic of conversation shifts, it can be a good approximation when the purpose of the system is limited (as it is in the UNIX Consultant). The implementor of such a system must take care to distinguish between goals that are *beyond* the range of goals that concern the system, and those that may be instrumental to goals that concern the system. The latter should be marked not as beyond the system's domain of expertise, but rather as instrumental goals.

As in determining when a goal is within the system's domain of expertise, determining instrumentality would ideally consider all the circumstances of the utterance, including knowledge of the speaker and of the previous dialogue. However, because of UC's limited domain, PAGAN can use its knowledge of the consulting process to make some assumptions about which goals are instrumental. Thus for example, linguistic goals are instrumental goals, because they are pursued only in order to achieve domain goals. On a more abstract level, any communicative goal is an instrumental goal for PAGAN. Consider the complaint:

User: I tried typing 'rm foo', but it told me that foo wasn't found.

In this example, the goal of typing is aimed at communicating with UNIX, and is therefore an instrumental goal. A goal of executing a program is also an instrumental goal. In this example, the goal of executing the rm command, which may be inferred from the

goal of typing 'rm,' is therefore also an instrumental goal.

Once the goal types contained in the system have been categorized according to instrumentality, these goals can be marked as such in long-term memory. Recognition of instrumental goals is then simply a matter of determining whether the goal under consideration bears the instrumental attribute. Thus, each goal that is included in PAGAN's knowledge base is described there either as an instrumental goal, a domain goal, or a goal that is beyond UC's domain. See the following section for a processing example that illustrates these points.

8.4.2. Identifying Strong Expectations

Strong expectations are identified by examining the dialogue representation. A strong expectation is indicated when a pending step of a plan is an action to be taken by the user. The strength of the expectation is a function of the recency of the immediately preceding plan step. If the preceding step has just been performed, the strength of the expectation is high. For example, immediately after the system asks a question of the user, there is a strong expectation that the user will respond to the question. On the other hand, if a larger number of utterances have occurred since the preceding plan step was executed, the strength of the expectation is low. This corresponds to the intuition that the longer an expectation remains unmet, the less likely it is to be met eventually.

Processing Example

The following trace shows PAGAN's processing of the user's utterance in the exchange:

> UC: Is the file in your directory?
> User: The file is in /tmp.

Only those portions of the trace relevant to PAGAN's decision about chaining are shown. In this example, the chaining decision is made three times. The first time, analysis is continued because the inferred goal is an instrumental goal, and the explanation chain is therefore incomplete. The second time, analysis is also continued, in this case because there is a strong expectation for a response to the question. The last time, analysis is halted because the user's utterance has been grounded in the preceding dialogue, and the explanation is therefore complete.

UC: Is the file in your directory?

User: The file is in /tmp.

Interpretation produced by the parser: TELL-ACTION36

The first chaining decision is based on the representation of the user's utterance. Since telling is considered an instrumental goal, chaining continues.

Determining whether to chain on TELL-ACTION36.
  Determining whether TELL-ACTION36 is a complete explanation of (THE FILE IS IN /TMP).
    Determining whether TELL-ACTION36 is outside of UC's domain of expertise.
    TELL-ACTION36 is not outside of UC's domain of expertise.
  The explanation is not necessarily complete.
  Determining whether TELL-ACTION36 is an incomplete explanation of
   (THE FILE IS IN /TMP).
    Determining whether TELL-ACTION36 is an instrumental goal.
    TELL-ACTION36 is an instrumental goal.
  The explanation is incomplete.
Chaining should be done on TELL-ACTION36.

Determining whether TELL-ACTION36 is grounded in the preceding dialogue.
Trying to ground TELL-ACTION36 in INDICATE-YES-OR-NO.
  Concreting TELL-ACTION36 to be INDICATE-YES-OR-NO.
TELL-ACTION36 was not grounded in INDICATE-YES-OR-NO because:
 Can't concrete UTTERANCE35 to UTTERANCE-OF-INDICATE-YES-OR-NO because the ranges
  differ.
Determining whether TELL-ACTION36 is a new plan for the existing goal of
 KNOWING-WHETHER13
TELL-ACTION36 is not a new plan for the existing goal of KNOWING-WHETHER13
 because:
  Attempt to concrete incompatibly from TELL-ACTION36 to KNOWING-WHETHER13.
TELL-ACTION36 was not grounded in the preceding dialogue.

Found PLANFOR3 as an explanation for TELL-ACTION36.
Synopsis:
 Goal: hearer know whether X;
 Step 1: Tell hearer that X.
Inferred goal is KNOWING-WHETHER14.

The second chaining decision is based on the user's goal of UC *knowing whether the file is in the directory. To demonstrate the use of strong expectations in chaining, I have removed the concept of* KNOWING *(from which* KNOWING-WHETHER14 *is descended) from the set of instrumental goals for this example. However, chaining is still performed, because* PAGAN *has a strong expectation that the user will respond to* UC*'s question (represented by* PLAN88*).*

Determining whether to chain on KNOWING-WHETHER14.
  Determining whether KNOWING-WHETHER14 is a complete explanation of TELL-ACTION36.
    Determining whether KNOWING-WHETHER14 is outside of UC's domain of expertise.

KNOWING-WHETHER14 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether KNOWING-WHETHER14 is an incomplete explanation of TELL-ACTION36.
Determining whether KNOWING-WHETHER14 is an instrumental goal.
KNOWING-WHETHER14 is not an instrumental goal.
Determining whether there is a strong expectation for a particular action.
There is a strong expectation for PLAN88
The explanation is incomplete.
Chaining should be done on KNOWING-WHETHER14.

Now, PAGAN *tries to determine whether* KNOWING-WHETHER14 *fits into the previous dialogue somehow. It does, so processing is halted.*

Determining whether KNOWING-WHETHER14 is grounded in the preceding dialogue.
Trying to ground KNOWING-WHETHER14 in INDICATE-YES-OR-NO.
KNOWING-WHETHER14 was not grounded in INDICATE-YES-OR-NO because:
Attempt to concrete incompatibly from KNOWING-WHETHER14 to INDICATE-YES-OR-NO.
Determining whether KNOWING-WHETHER14 is a new plan for the existing goal of KNOWING-WHETHER13
KNOWING-WHETHER14 is a new plan for the existing goal of KNOWING-WHETHER13
Existing plan PLAN87 has been rejected; new plan is PLAN89.

8.4.3. Determining Difficulty of Analysis

Part of the task of determining difficulty of analysis is subsumed by the task of determining whether a goal is an instrumental goal, a domain goal, or a goal outside of the system's domain. PAGAN assumes that instrumental goals are easy to explain, but that goals outside of the system's domain are quite difficult to explain (since it is unlikely that the system contains the requisite knowledge). Thus, the only goals for which difficulty of analysis must be assessed are domain goals.

Determining the difficulty of analysis for domain goals requires a method to estimate resource usage for continuation of a particular explanation chain. The only way to know this cost for sure is to carry out the analysis. Consequently, a heuristic approach to the problem is called for. One of the simplest such heuristics is to estimate the cost of continuing an explanation chain in proportion to the cost incurred so far by that explanation chain. This can be done by monitoring the total branching that has taken place among all explanation chains of the utterance that are under consideration. Once this amount exceeds a threshold, any further branching brings the principle of difficulty of analysis into play. The strength of the principle is proportional to the total amount of branching found among all active explanation chains. This is the approach taken in the implementation of PAGAN.

8.4.4. Determining Curiosity

The interestingness of a goal is dictated by the relationships that exist between that goal and other goals held by the system and by the user. Wilensky [1978] describes the detection of goal relationships in the context of story understanding. Often, it is possible to short-cut the process of detecting goal relationships by marking certain concepts as inherently interesting. For example, each of the following questions addresses a goal that might be inherently interesting to UC:

> User: How can I get the sources to Rogue?
>
> User: How can I lay waste to a directory?
>
> User: How do I change a.out?

The first example concerns an interesting object: the game of Rogue. The second addresses an interesting action: wreaking havoc. Each of these goals might well be considered inherently interesting by a UNIX Consultant system. The third example addresses the goal of manipulating the contents of a binary file. Neither of the concepts in this example, 'changing a file' and 'binary file,' is interesting in itself. Rather, it is the combination of the two that produces a concept that is interesting. Note that the decision about whether a goal is inherently interesting is not a decision that may be made by the goal analyzer without considering the task of the system in which it is embedded; rather, the criteria for what deserves automatic attention will change as the domain in which the goal analyzer operates changes.

Processing Example

The following trace shows the chaining decisions made by PAGAN in processing the utterance:

> User: How could I change a.out?

This trace illustrates the use of curiosity. PAGAN is curious about the goal of changing a binary file. PAGAN therefore chains on the user's goal of changing the binary file, inferring the goal of changing the effect of the program.

User: How could I change a.out?

Interpretation produced by the parser: TELL-ACTION38

> *The first three goals inferred by* PAGAN *are instrumental goals, so chaining continues from each of them.*

Determining whether to chain on TELL-ACTION38.

Determining whether TELL-ACTION38 is a complete explanation of

(HOW COULD I CHANGE A OUT).
Determining whether TELL-ACTION38 is outside of UC's domain of expertise.
TELL-ACTION38 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether TELL-ACTION38 is an incomplete explanation of
(HOW COULD I CHANGE A OUT).
Determining whether TELL-ACTION38 is an instrumental goal.
TELL-ACTION38 is an instrumental goal.
The explanation is incomplete.
Chaining should be done on TELL-ACTION38.

Determining whether TELL-ACTION38 is grounded in the preceding dialogue.
TELL-ACTION38 was not grounded in the preceding dialogue.

Found PLANFOR4 as an explanation for TELL-ACTION38.
Synopsis:
Goal: know something;
Step 1: ask;
Step 2: be told.
Inferred goal is KNOWING-THAT15.

Determining whether to chain on KNOWING-THAT15.
Determining whether KNOWING-THAT15 is a complete explanation of TELL-ACTION38.
Determining whether KNOWING-THAT15 is outside of UC's domain of expertise.
KNOWING-THAT15 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether KNOWING-THAT15 is an incomplete explanation of TELL-ACTION38.
Determining whether KNOWING-THAT15 is an instrumental goal.
KNOWING-THAT15 is an instrumental goal.
The explanation is incomplete.
Chaining should be done on KNOWING-THAT15.

Determining whether KNOWING-THAT15 is grounded in the preceding dialogue.
KNOWING-THAT15 was not grounded in the preceding dialogue.

Found PLANFOR8 as an explanation for KNOWING-THAT15.
Synopsis:
Goal: Achieve X;
Step 1: Know how to X;
Step 2: Do X
Inferred goal is CHANGE-EVENT3.

The next goal, CHANGE-EVENT3 *(which represents changing the file a.out), is not an instrumental goal, nor is there a strong expectation for a particular action. However, the goal of changing an executable file arouses interest in* PAGAN*; thus, chaining continues.*

Determining whether to chain on CHANGE-EVENT3.

Determining whether CHANGE-EVENT3 is a complete explanation of KNOWING-THAT15.
Determining whether CHANGE-EVENT3 is outside of UC's domain of expertise.
CHANGE-EVENT3 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether CHANGE-EVENT3 is an incomplete explanation of KNOWING-THAT15.
Determining whether CHANGE-EVENT3 is an instrumental goal.
CHANGE-EVENT3 is not an instrumental goal.
Determining whether there is a strong expectation for a particular action.
There is no strong expectation for a particular action.
The explanation is not necessarily incomplete.
Determining the level of curiosity about CHANGE-EVENT3.
Determining the level of interest in CHANGE-EVENT3.
Interest level = 4.
Determining the level to which the goal CHANGE-EVENT3 conflicts with system goals.
Conflict level = 0.
Curiosity level = 4.
Determining estimated cost of chaining on CHANGE-EVENT3.
Estimated cost = 1.
Chaining should be done on CHANGE-EVENT3.

Determining whether CHANGE-EVENT3 is grounded in the preceding dialogue.
CHANGE-EVENT3 was not grounded in the preceding dialogue.

Found PLANFOR9 as an explanation for CHANGE-EVENT3.
Synopsis:
Goal: change program effect;
Step 1: Change the program's binary file
Inferred goal is CHANGE-PROGRAM-EFFECT0.

The newly-inferred goal is the goal of changing the effects of the program. This goal does not arouse curiosity in PAGAN, *so chaining stops.*

Determining whether to chain on CHANGE-PROGRAM-EFFECT0.
Determining whether CHANGE-PROGRAM-EFFECT0 is a complete explanation of CHANGE-EVENT3.
Determining whether CHANGE-PROGRAM-EFFECT0 is outside of UC's domain of expertise.
CHANGE-PROGRAM-EFFECT0 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether CHANGE-PROGRAM-EFFECT0 is an incomplete explanation of CHANGE-EVENT3.
Determining whether CHANGE-PROGRAM-EFFECT0 is an instrumental goal.
CHANGE-PROGRAM-EFFECT0 is not an instrumental goal.
Determining whether there is a strong expectation for a particular action.
There is no strong expectation for a particular action.
The explanation is not necessarily incomplete.
Determining the level of curiosity about CHANGE-PROGRAM-EFFECT0.
Determining the level of interest in CHANGE-PROGRAM-EFFECT0.

Interest level = 0.
Determining the level to which the goal CHANGE-PROGRAM-EFFECT0 conflicts with system goals.
Conflict level = 0.
Curiosity level = 0.
Determining estimated cost of chaining on CHANGE-PROGRAM-EFFECT0.
Estimated cost = 1.
Chaining should not be done on CHANGE-PROGRAM-EFFECT0.

8.5. Summary

Once it infers the motivating goal of an utterance, a goal analyzer must determine whether that goal should itself be explained. A variety of methods for making this determination have been put forward in the literature. In this chapter, I attempted to place those methods that seem justifiable into a unified framework. This framework is centered on a determination of whether the explanation is *complete*, as described in Chapter 3. If the explanation is complete, chaining is stopped. If the explanation is incomplete, then goal analysis is continued. However, if it cannot be determined whether the explanation is complete, then the system must weigh the strength of its interest in continuing the analysis against the estimated cost of doing so.

Chapter 9

Implementation and Extended Example

9.1. Implementation of KODIAK

Many of the ideas described herein rely for their implementation on aspects of the KODIAK knowledge representation system. KODIAK has been described extensively by Wilensky [1987a] and by Norvig [1987]. I will describe here only differences between PAGAN's version of KODIAK and these previous approaches.

A significant point of difference from previous implementations is that PAGAN's implementation of KODIAK allows the parallel development of rival knowledge structures, that is, of multiple potentially mutually exclusive representations. For example, it allows two potential explanations of an utterance to co-exist in the knowledge base. These rival structures do not interact; they are completely isolated from one another. At any point in PAGAN's processing, a branch-point may be introduced. Each branch has available to it all of the knowledge introduced before the branching. However, knowledge introduced after the branching is visible only in the branch that created it. Once a particular branch has been selected as the correct one, it can be made the basis for all future inferences. However, it is always possible to go back to a branch that was previously rejected and continue its development.

The use of a branching mechanism eliminates the need for backup, and greatly reduces the need for a non-monotonic abductive reasoning mechanism. This is because competing knowledge structures can be developed in parallel, and the decision as to which structure to trust can be delayed as long as necessary.

Parallel development is achieved through the use of keyed knowledge. Each fact in the knowledge base has an associated key. These keys are hierarchically organized according to the branching structure that has been introduced by PAGAN (or by some other component of the system). When retrieving facts from the knowledge base, a key is provided; only facts that are keyed with this key or one of its ancestors are returned.

For example, suppose that PAGAN is entertaining two rival explanations for utterance U, called A and B. Furthermore, suppose that A is itself explained either by X or by Y, and B is explained by Z. In this example then, there are two branch-points: one at U, and the other at A. A picture of these relationships is shown in Figure 9.1. At the left of the picture is the hierarchy of sets of KODIAK objects that compose the explanations, and at the right is the hierarchy of keys associated with each set.

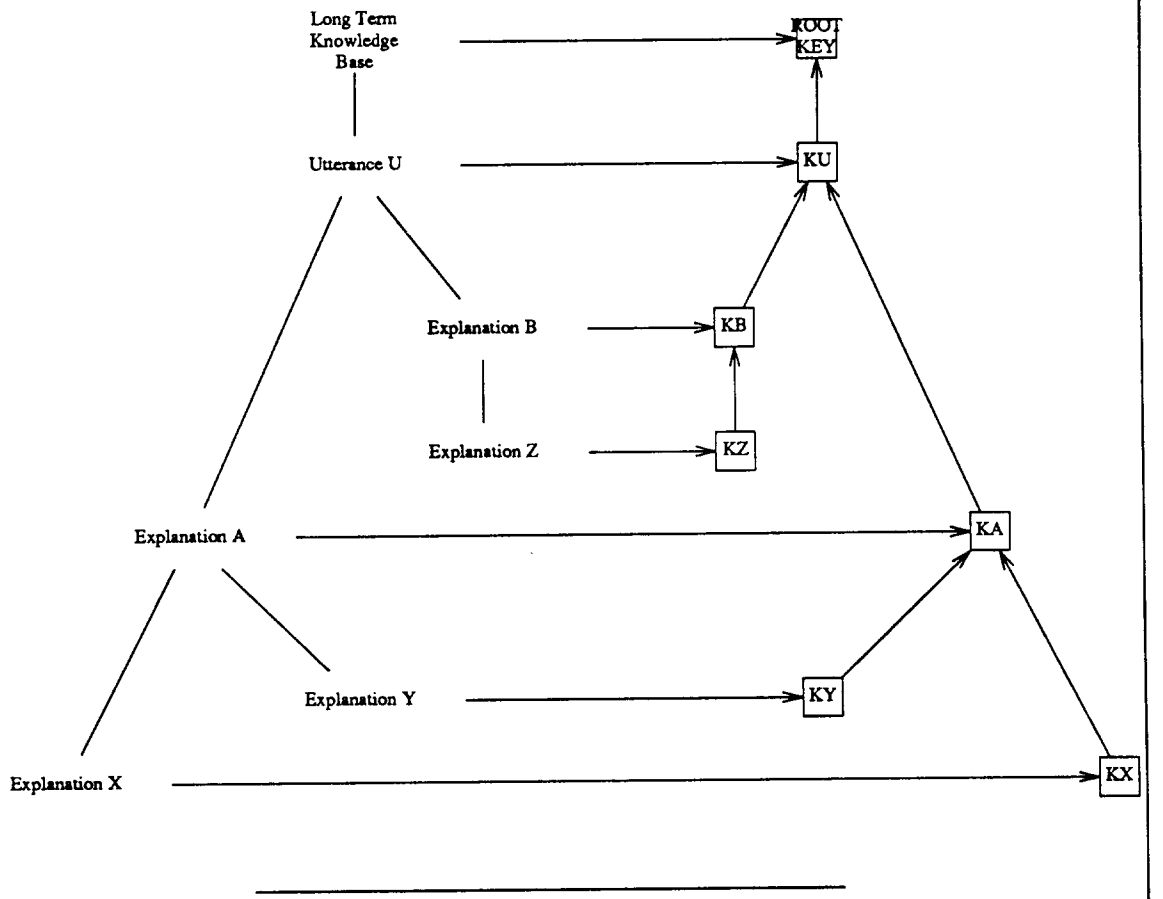

Figure 9.1. A Hierarchy of Explanations and their Keys

When PAGAN needs to retrieve a fact from the knowledge base, it provides KODIAK with a key. In searching for the fact, KODIAK examines only knowledge based on that key or one of that key's ancestors. For example, suppose that in evaluating explanation Y, PAGAN needs to collect all the relations in which one of the components (call it C) of explanation A takes part. Since it is examining explanation Y, PAGAN uses KY as the search key. Although C is part of explanation A, and therefore has been created with key KA, it nonetheless may take part in relations with keys KX and KY. Since key KY has been provided as the search key though, only relations built on key KY or one of its ancestors will be retrieved. That is, the relations returned by KODIAK will be composed entirely of concepts that are part of the long-term knowledge base, the representation of utterance U, explanation A, and explanation Y. Thus, explanation X and explanation B are effectively invisible to PAGAN when it is using key KY.

9.2. System Architecture and Performance

PAGAN was developed in conjunction with a parser, and the version of KODIAK described above. The parser, based on the ALANA parser [Cox 1986], is a bottom-up chart-based parser. Associated with each grammar rule is a Lisp function that builds the semantics of the left-hand side of the rule out of the semantics of the components of its right-hand side. These functions are applied only after the input sentence has been completely parsed. If there are multiple syntactic parses, a separate key (as described above) is used in building the semantics for each of them.

The processes of concretion and equation (see Chapter 5) are built into the KODIAK interpreter. As each new concept is entered into the knowledge base by the parser, these processes are automatically invoked. Thus, much of the categorization required by PAGAN has already been completed by the time PAGAN is given the representation of the user's utterance.

PAGAN's operation proceeds as described in the preceding chapters. During its processing, PAGAN builds a model of the dialogue. This model contains representations of each of the plans and goals that PAGAN infers to explain the user's utterances. This network of plans and goals is PAGAN's output. It is up to the remainder of the system to decide which of these goals should be addressed, and which of these plans should be continued. See Chin [1988] for a description of how UC makes these decisions.

PAGAN is implemented in Common Lisp with Explorer extensions on a TI Explorer II Lisp machine. It occupies about 3000 lines of code. The initial knowledge base contains approximately 500 KODIAK concepts. Processing of an average utterance (such as a single question) from parsing through goal analysis by PAGAN takes approximately 40 seconds. The bulk of this time is spent traversing the KODIAK network; speed has largely been sacrificed for safety and convenience.

9.3. Detailed Trace

In this section, I present a detailed trace of PAGAN's processing of the exchange:

> UC: Is the file in your directory?
> User: It is in /tmp.

I have used this and similar examples throughout this dissertation to demonstrate various points. Here I provide the entire trace of the processing of this exchange. PAGAN's processing of UC's question and of the user's response are both included.

UC: Is the file in your directory?

This portion of the trace reflects PAGAN*'s processing of* UC*'s own question. First, the question is parsed, and a semantic interpretation (*TELL-ACTION43*) is produced.*

Applying semantics for TO-BE => IS
Applying semantics for ARTICLE => THE
Applying semantics for NOMINAL => FILE
Applying semantics for NP2 => ARTICLE NOMINAL
Applying semantics for NP => NP2
Applying semantics for PREPOSITION => IN
Applying semantics for POSSESSIVE => YOUR
Applying semantics for NOMINAL => DIRECTORY
Applying semantics for NP2 => POSSESSIVE NOMINAL
Applying semantics for NP => NP2
Applying semantics for PP => PREPOSITION NP
Applying semantics for SENTENCE => TO-BE NP PP

Here, the speaker of the utterance is equated with the actor of the utterance (which has already been set to UC*).*

Equating the range of SPEAKER39 (which is ANIMATE)
with the range of ACTOR58 (which is UNIX-CONSULTANT).

Because the thing told by UC *is a verification question,* TELL-ACTION43 *is automatically concreted first to an* ASK-QUESTION*, then to an* ASK-VERIFICATION-QUESTION.

Concreting TELL-ACTION43 to be ASK-QUESTION.
Concreting UTTERANCE42 to be UTTERANCE-OF-ASK-QUESTION.
Concreting TELL-ACTION43 to be ASK-VERIFICATION-QUESTION.
Concreting UTTERANCE42 to be UTTERANCE-OF-ASK-VERIFICATION-QUESTION.

Here is the representation of UC*'s utterance produced by the parser.*

The parser produced the following interpretation:

Interpretation 1 (TELL-ACTION43):
(HAS-GOAL106 (DOMAIN UNIX-CONSULTANT)
 (RANGE
  (TELL-ACTION43 (DOMINATED BY ASK-VERIFICATION-QUESTION)
   (ACTOR58 = UNIX-CONSULTANT) (HEARER40 = USER)
   (SPEAKER39 = UNIX-CONSULTANT)
   (UTTERANCE42 =
    (VERIFICATION-QUESTION15 (DOMINATED BY VERIFICATION-QUESTION)
     (WHAT-IS31 =
      (LOCATION-OF10 (DOMINATED BY LOCATION-OF)
       (LOCATED10 = (FILE24 (DOMINATED BY FILE)))
       (LOCATION10 =
         (DIRECTORY6 (DOMINATED BY DIRECTORY) (OWNS9* = USER)))))))))

First, PAGAN *checks to make sure that there is no conflict between this interpretation and the user model.*

Attempting to resolve any ambiguities in TELL-ACTION43
  Retrieving potential conflicts with TELL-ACTION43 and its plan.
  No conflict found.

Next, PAGAN *determines whether any further analysis is necessary. It decides that the goal of asking the question is an instrumental goal, and therefore that processing should go forward.*

Determining whether to chain on TELL-ACTION43.
  Determining whether TELL-ACTION43 is a complete explanation of
   (IS THE FILE IN YOUR DIRECTORY).
    Determining whether TELL-ACTION43 is outside of UC's domain of expertise.
    TELL-ACTION43 is not outside of UC's domain of expertise.
  The explanation is not necessarily complete.
  Determining whether TELL-ACTION43 is an incomplete explanation of
   (IS THE FILE IN YOUR DIRECTORY).
    Determining whether TELL-ACTION43 is an instrumental goal.
    TELL-ACTION43 is an instrumental goal.
  The explanation is incomplete.
Chaining should be done on TELL-ACTION43.

PAGAN *now tries to find an explanation for the question. It starts by determining whether the question is grounded in the preceding dialogue. Since in this example there is no previous dialogue, no grounding is found.*

Attempting to find an explanation for TELL-ACTION43.
  Determining whether TELL-ACTION43 is grounded in the preceding dialogue.
  TELL-ACTION43 was not grounded in the preceding dialogue.

An explanation is now sought among the planfors in long-term memory. An applicable planfor is found. This planfor states that to know whether a condition holds, ask whether it holds and receive a reply. PAGAN *incorporates a copy of this planfor into the dialogue model, automatically making the appropriate concretions in the process.*

  Trying to find planfor explanations for TELL-ACTION43.
  Found PLANFOR2 as an explanation for TELL-ACTION43.
  Synopsis:
   Goal: Know whether f(X) = Y;
   Step 1: Ask whether f(X) = Y;
   Step 2: Hearer says yes or no.
  Equating the range of KNOWN40 (which is THING)
   with the range of WHAT-IS31 (which is LOCATION-OF10).
  Equating the range of KNOWER40 (which is ANIMATE)
   with the range of SPEAKER39 (which is UNIX-CONSULTANT).
  Inferred goal is KNOWING-WHETHER19.

Explanation found for TELL-ACTION43.

Again, PAGAN *verifies that there is no conflict between the newly-inferred goal and the user model.*

Attempting to resolve any ambiguities in KNOWING-WHETHER19
Retrieving potential conflicts with KNOWING-WHETHER19 and its plan.
No conflict found.

PAGAN *must now determine whether to chain on the new goal. Once again, the goal is an instrumental goal, so processing continues.*

Determining whether to chain on KNOWING-WHETHER19.
Determining whether KNOWING-WHETHER19 is a complete explanation of TELL-ACTION43.
Determining whether KNOWING-WHETHER19 is outside of UC's domain of expertise.
KNOWING-WHETHER19 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether KNOWING-WHETHER19 is an incomplete explanation of TELL-ACTION43.
Determining whether KNOWING-WHETHER19 is an instrumental goal.
KNOWING-WHETHER19 is an instrumental goal.
The explanation is incomplete.
Chaining should be done on KNOWING-WHETHER19.

An explanation is now sought for the newly-inferred goal. Lacking any previous dialogue, no grounding can be performed.

Attempting to find an explanation for KNOWING-WHETHER19.
Determining whether KNOWING-WHETHER19 is grounded in the preceding dialogue.
KNOWING-WHETHER19 was not grounded in the preceding dialogue.

A planfor explanation is found. This planfor states, roughly, that to know something, know that it has a particular value. Again, the appropriate equations are made when the dialogue structure is augmented.

Trying to find planfor explanations for KNOWING-WHETHER19.
Found PLANFOR1 as an explanation for KNOWING-WHETHER19.
Synopsis:
Goal: Know f(X);
Step 1: Know that f(X) = Y
Equating the range of KNOWN41 (which is THING)
with the range of KNOWN40 (which is LOCATION-OF10).
Equating the range of KNOWER41 (which is ANIMATE)
with the range of KNOWER40 (which is UNIX-CONSULTANT).
Inferred goal is KNOWING-THAT20.

Explanation found for KNOWING-WHETHER19.

No conflicts are found between the newly-inferred goal and the user model.

Attempting to resolve any ambiguities in KNOWING-THAT20
Retrieving potential conflicts with KNOWING-THAT20 and its plan.

No conflict found.

PAGAN *decides that chaining should be done on the new goal, since it is an instrumental goal.*

Determining whether to chain on KNOWING-THAT20.
Determining whether KNOWING-THAT20 is a complete explanation of KNOWING-WHETHER19.
Determining whether KNOWING-THAT20 is outside of UC's domain of expertise.
KNOWING-THAT20 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether KNOWING-THAT20 is an incomplete explanation of KNOWING-WHETHER19.
Determining whether KNOWING-THAT20 is an instrumental goal.
KNOWING-THAT20 is an instrumental goal.
The explanation is incomplete.
Chaining should be done on KNOWING-THAT20.

I have not supplied PAGAN *with any knowledge about why* UC *might want to know the location of a file, so* PAGAN*'s processing of* UC*'s utterance stops at this point.*

Attempting to find an explanation for KNOWING-THAT20.
Determining whether KNOWING-THAT20 is grounded in the preceding dialogue.
KNOWING-THAT20 was not grounded in the preceding dialogue.

Trying to find planfor explanations for KNOWING-THAT20.
No planfor explanations found for KNOWING-THAT20.

Determining whether KNOWING-THAT20 is explained by a theme.
KNOWING-THAT20 was not explained by a theme.

No explanation found for KNOWING-THAT20.

Next, the user's response is analyzed. Since the parser is unable to determine the referent of 'it,' two interpretations are produced.

User: It is in /tmp.

Applying semantics for NP => IT
Applying semantics for TO-BE => IS
Applying semantics for PREPOSITION => IN
Applying semantics for NP => /TMP
Applying semantics for PP => PREPOSITION NP
Applying semantics for SENTENCE => NP TO-BE PP
Equating the range of SPEAKER40 (which is ANIMATE)
with the range of ACTOR59 (which is USER).
Concreting TELL-ACTION44 to be MAKE-STATEMENT.

Applying semantics for NP => IT
Applying semantics for TO-BE => IS
Applying semantics for PREPOSITION => IN
Applying semantics for NP => /TMP
Applying semantics for PP => PREPOSITION NP
Applying semantics for SENTENCE => NP TO-BE PP
Equating the range of SPEAKER41 (which is ANIMATE)
with the range of ACTOR60 (which is USER).
Concreting TELL-ACTION45 to be MAKE-STATEMENT.

The parser produced the following interpretations:

In the first interpretation, the user is indicating the location of the file.

Interpretation 1 (TELL-ACTION44):
(HAS-GOAL109 (DOMAIN USER)
(RANGE
(TELL-ACTION44 (DOMINATED BY MAKE-STATEMENT) (ACTOR59 = USER)
(HEARER41 = UNIX-CONSULTANT) (SPEAKER40 = USER)
(UTTERANCE43 =
(LOCATION-OF11 (DOMINATED BY LOCATION-OF)
(LOCATED11 = (FILE25 (DOMINATED BY FILE)))
(LOCATION11 = SLASH-TMP))))))

In the second interpretation, the user is indicating the location of the directory.

Interpretation 2 (TELL-ACTION45):
(HAS-GOAL110 (DOMAIN USER)
(RANGE
(TELL-ACTION45 (DOMINATED BY MAKE-STATEMENT) (ACTOR60 = USER)
(HEARER42 = UNIX-CONSULTANT) (SPEAKER41 = USER)
(UTTERANCE44 =
(LOCATION-OF12 (DOMINATED BY LOCATION-OF)
(LOCATED12 = (DIRECTORY7 (DOMINATED BY DIRECTORY)))
(LOCATION12 = SLASH-TMP))))))

Attempting to resolve any ambiguities in TELL-ACTION45
Retrieving potential conflicts with TELL-ACTION45 and its plan.
No conflict found.

Since this is an instrumental goal, processing continues on the 'directory' interpretation.

Determining whether to chain on TELL-ACTION45.
Determining whether TELL-ACTION45 is a complete explanation of (IT IS IN /TMP).
Determining whether TELL-ACTION45 is outside of UC's domain of expertise.
TELL-ACTION45 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether TELL-ACTION45 is an incomplete explanation of (IT IS IN /TMP).
Determining whether TELL-ACTION45 is an instrumental goal.

TELL-ACTION45 is an instrumental goal.
The explanation is incomplete.
Chaining should be done on TELL-ACTION45.

Attempting to resolve any ambiguities in TELL-ACTION44
Retrieving potential conflicts with TELL-ACTION44 and its plan.
No conflict found.

Chaining should also be done on the 'file' interpretation.

Determining whether to chain on TELL-ACTION44.
Determining whether TELL-ACTION44 is a complete explanation of (IT IS IN /TMP).
Determining whether TELL-ACTION44 is outside of UC's domain of expertise.
TELL-ACTION44 is not outside of UC's domain of expertise.
The explanation is not necessarily complete.
Determining whether TELL-ACTION44 is an incomplete explanation of (IT IS IN /TMP).
Determining whether TELL-ACTION44 is an instrumental goal.
TELL-ACTION44 is an instrumental goal.
The explanation is incomplete.
Chaining should be done on TELL-ACTION44.

PAGAN *now looks for an explanation of the 'file' interpretation. Since there has been previous dialogue, grounding is more complex than in the processing of the previous utterance.* PAGAN *must check both to see whether this goal matches an expectation, and whether it represents a new plan for an existing goal. Neither condition holds, so the 'file' interpretation has not yet been grounded.*

Attempting to find an explanation for TELL-ACTION44.
Determining whether TELL-ACTION44 is grounded in the preceding dialogue.
Trying to ground TELL-ACTION44 in INDICATE-YES-OR-NO.
Concreting TELL-ACTION44 to be INDICATE-YES-OR-NO.
TELL-ACTION44 was not grounded in INDICATE-YES-OR-NO because:
Can't concrete UTTERANCE43 to UTTERANCE-OF-INDICATE-YES-OR-NO because the ranges differ.
Determining whether TELL-ACTION44 is a new plan for the existing goal of KNOWING-WHETHER19
TELL-ACTION44 is not a new plan for the existing goal of KNOWING-WHETHER19 because:
Attempt to concrete incompatibly from TELL-ACTION44 to KNOWING-WHETHER19.
TELL-ACTION44 was not grounded in the preceding dialogue.

Now, PAGAN *tries to find a planfor explanation for the utterance under the 'file' interpretation. It finds an applicable planfor, which states that telling something to someone is a plan for the goal of that person knowing whether it is true.*

Trying to find planfor explanations for TELL-ACTION44.
Found PLANFOR3 as an explanation for TELL-ACTION44.
Synopsis:
Goal: hearer know whether X;

Step 1: Tell hearer that X.
Equating the range of KNOWN42 (which is THING)
with the range of UTTERANCE43 (which is LOCATION-OF11).
Equating the range of KNOWER42 (which is ANIMATE)
with the range of HEARER41 (which is UNIX-CONSULTANT).
Inferred goal is KNOWING-WHETHER20.

Explanation found for TELL-ACTION44.

An explanation is also sought for the 'directory' interpretation. This processing exactly parallels the processing done for the 'file' interpretation above.

Attempting to find an explanation for TELL-ACTION45.
Determining whether TELL-ACTION45 is grounded in the preceding dialogue.
Trying to ground TELL-ACTION45 in INDICATE-YES-OR-NO.
Concreting TELL-ACTION45 to be INDICATE-YES-OR-NO.
TELL-ACTION45 was not grounded in INDICATE-YES-OR-NO because:
Can't concrete UTTERANCE44 to UTTERANCE-OF-INDICATE-YES-OR-NO because the ranges differ.
Determining whether TELL-ACTION45 is a new plan for the existing goal of KNOWING-WHETHER19
TELL-ACTION45 is not a new plan for the existing goal of KNOWING-WHETHER19 because:
Attempt to concrete incompatibly from TELL-ACTION45 to KNOWING-WHETHER19.
TELL-ACTION45 was not grounded in the preceding dialogue.

Trying to find planfor explanations for TELL-ACTION45.
Found PLANFOR3 as an explanation for TELL-ACTION45.
Synopsis:
Goal: hearer know whether X;
Step 1: Tell hearer that X.
Equating the range of KNOWN43 (which is THING)
with the range of UTTERANCE44 (which is LOCATION-OF12).
Equating the range of KNOWER43 (which is ANIMATE)
with the range of HEARER42 (which is UNIX-CONSULTANT).
Inferred goal is KNOWING-WHETHER21.

Explanation found for TELL-ACTION45.

PAGAN *now examines the 'directory' interpretation to determine whether the ambiguity can be resolved. However, no conflict is found with the user model. Of course, with a more knowledgable user model, the ambiguity might be resolved here.*

Attempting to resolve any ambiguities in KNOWING-WHETHER21
Retrieving potential conflicts with KNOWING-WHETHER21 and its plan.
No conflict found.

Chaining should be performed on the 'directory' interpretation, because the newly-inferred goal is an instrumental goal.

Determining whether to chain on KNOWING-WHETHER21.
  Determining whether KNOWING-WHETHER21 is a complete explanation of TELL-ACTION45.
    Determining whether KNOWING-WHETHER21 is outside of UC's domain of expertise.
    KNOWING-WHETHER21 is not outside of UC's domain of expertise.
  The explanation is not necessarily complete.
  Determining whether KNOWING-WHETHER21 is an incomplete explanation of TELL-ACTION45.
    Determining whether KNOWING-WHETHER21 is an instrumental goal.
    KNOWING-WHETHER21 is an instrumental goal.
  The explanation is incomplete.
Chaining should be done on KNOWING-WHETHER21.

The same processing is done on the 'file' interpretation.

Attempting to resolve any ambiguities in KNOWING-WHETHER20
  Retrieving potential conflicts with KNOWING-WHETHER20 and its plan.
  No conflict found.

Determining whether to chain on KNOWING-WHETHER20.
  Determining whether KNOWING-WHETHER20 is a complete explanation of TELL-ACTION44.
    Determining whether KNOWING-WHETHER20 is outside of UC's domain of expertise.
    KNOWING-WHETHER20 is not outside of UC's domain of expertise.
  The explanation is not necessarily complete.
  Determining whether KNOWING-WHETHER20 is an incomplete explanation of TELL-ACTION44.
    Determining whether KNOWING-WHETHER20 is an instrumental goal.
    KNOWING-WHETHER20 is an instrumental goal.
  The explanation is incomplete.
Chaining should be done on KNOWING-WHETHER20.

Now, PAGAN *tries to explain why* UC *might want to know the location of the file. First, it checks to see whether this goal satisfies an expectation set up by the previous utterance.*

Attempting to find an explanation for KNOWING-WHETHER20.
  Determining whether KNOWING-WHETHER20 is grounded in the preceding dialogue.
  Trying to ground KNOWING-WHETHER20 in INDICATE-YES-OR-NO.
  KNOWING-WHETHER20 was not grounded in INDICATE-YES-OR-NO because:
    Attempt to concrete incompatibly from KNOWING-WHETHER20 to INDICATE-YES-OR-NO.

Next, PAGAN *determines whether this goal represents a new plan for* UC*'s goal. This time it is successful, and* UC*'s old plan is discarded.*

  Determining whether KNOWING-WHETHER20 is a new plan for the existing goal of KNOWING-WHETHER19
    Concreting KNOWING-WHETHER20 to be KNOWING-WHETHER19.
  KNOWING-WHETHER20 is a new plan for the existing goal of KNOWING-WHETHER19

Existing plan PLAN106 has been rejected; new plan is PLAN109.

The entire chain of plans and goals that underlie KNOWING-WHETHER20 *is selected as the correct interpretation of the user's utterance. The competing explanations, derived from the interpretation of 'it' as 'directory,' are rejected. This completes* PAGAN*'s processing.*

Selecting KNOWING-WHETHER20 as the correct explanation of TELL-ACTION44
Selecting TELL-ACTION44 as the correct explanation of (IT IS IN /TMP)
Rejecting TELL-ACTION45 as an explanation of (IT IS IN /TMP).
Rejecting KNOWING-WHETHER21 as an explanation of TELL-ACTION45.
Explanation found for KNOWING-WHETHER20.

Chapter 10

Conclusions

Artificial Intelligence programs tend to be hacks. In many ways, **PAGAN** is not an exception to this rule. One way to introduce a degree of theoretical grounding into an AI system is to provide a method for evaluating its output. The criteria listed in Chapter 3 are an attempt to address this issue. They have proved their usefulness by guiding **PAGAN**'s development. To the extent that they accurately reflect what it means to be a good explanation, they have helped to give **PAGAN** some theoretical grounding.

The biggest problem facing plan recognition systems is ambiguity. The number of ways that a particular sentence might be viewed as ambiguous is striking. Yet people rarely view a normal utterance as ambiguous. Ambiguity is resolved through recourse to additional knowledge held by the system that was not taken into account in producing the competing interpretations. The two most important sources of such knowledge are the user model and the dialogue model. User models in particular will become increasingly important to plan recognizers in the future.

The adoption of an appropriate representation is an important step in restricting ambiguity. *Intended* recognition should be modeled by representations that capture that intentionality. Traditional representations have included all of the effects of a plan. In this thesis, I have argued that a representation such as a planfor that focuses on *the* effect that is typically intended should be used for plan recognition.

The most difficult part of the *implementation* of **PAGAN** was the specification of world knowledge. **KODIAK** provides a flexible tool for representation, and provides a number of useful built-in inference mechanisms. However, it lends very little guidance as to *what* should be represented (in the way that Conceptual Dependency [Schank and Abelson 1977] did for example). In other words, while the **KODIAK** interpreter is itself quite robust, the representations contained in the knowledge base are not. Many of the bugs that arose during the implementation of **PAGAN** were not problems with the code, but rather problems with the knowledge base. These problems were of two main classes. First, it was fairly easy to use existing concepts incorrectly. Secondly, it was often unclear exactly how new knowledge should be expressed. The former problem can be

ameliorated through the introduction of more sophisticated tools for encoding knowledge than were available for the development of PAGAN. The latter problem is less easily dealt with. During the development of PAGAN, I would often find that two or more iterations were required before a particular concept was adequately represented. I expect this problem to remain as long as knowledge must be hand-coded, and must remain immutable once it has been entered.

This suggests two ways around the problem of knowledge encoding. First, new knowledge could be *automatically* generated and entered into the knowledge base. For example, explanation-based learning provides some hope that planfor knowledge could be automatically generated. See Braverman and Russell [1988] for a discussion of the problems in automatically creating planning knowledge.

The second way that this problem might be dealt with is by allowing knowledge entered into the system to dynamically rearrange itself. The person entering knowledge could then be relatively sloppy about the representations chosen; the system would be responsible for rearranging the initial representations in some coherent fashion. I expect that one or both of these approaches will be required by future large-scale plan recognition systems.

One final note is in order on the future of plan recognition systems. Current systems almost invariably attempt to answer the question "Why did the speaker say X?" I suspect that to become truly proficient at their task, plan recognition systems in the future will have to address not only this question, but also the question "Why *didn't* the speaker say Y?" As in bridge bidding, answering this latter question may provide far more information than answering the former question alone.

Plan recognition, dialogue modeling, and user modeling can all benefit from such an approach. For example, the fact that a user did not use the standard term for a particular concept might indicate to the user model that the user does not know that term, and is likely to be unfamiliar with the concept in general. This conclusion could not be supported solely by an analysis of what the user *did* say. Or, the use of a noun where a pronoun would have been perfectly acceptable might indicate to a dialogue model that there is a discrepency between the system's understanding of the dialogue and the user's understanding of the dialogue. Such a discrepency could not be detected simply by considering what the user *did* say, because the use of the noun conveys the same semantic information as the use of the appropriate pronoun. Finally, expectation matching is an example of how this approach is already useful in goal analysis. When the user's utterance does not conform to an expectation held by the system, what was *not* said can be just as important to PAGAN's processing of the utterance as what was said.

References

James F. Allen. *A Plan-Based Approach to Speech Act Recognition.* Doctoral dissertation. Technical Report 131/79, Department of Computer Science, University of Toronto, 1979.

James F. Allen and C. Raymond Perrault. Analyzing intention in utterances. *Artificial Intelligence* 15(3):143-178, 1980.

James F. Allen, Alan M. Frisch, and Diane J. Litman. ARGOT: the Rochester dialogue system. In *Proceedings of the National Conference on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann Publishers, 1982.

Douglas E. Appelt. Planning English referring expressions. *Artificial Intelligence* 26(1):1-33, 1985.

J. L. Austin. *How To Do Things With Words.* Second edition. Harvard University Press, 1962.

Lawrence Birnbaum. Lexical ambiguity as a touchstone for theories of language analysis. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann Publishers, 1985.

Daniel G. Bobrow, Ronald M. Kaplan, Martin Kay, Donald A. Norman, Henry Thompson, and Terry Winograd. GUS, a frame-driven dialog system. *Artificial Intelligence* 8(2):155-173, 1977.

Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science* 9(2):171-216, 1985

Michael S. Braverman and Stuart J. Russell. IMEX: Overcoming intractability in explanation based learning. In *Proceedings of the National Conference on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann Publishers, 1988.

Sandra Carberry. Tracking user goals in an information-seeking environment. In *Proceedings of the National Conference on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann Publishers, 1983.

Sandra Carberry. TRACK: Toward a robust natural language interface. In *Proceedings of the Canadian National Conference on Artificial Intelligence*, 1986a.

Sandra Carberry. User models: The problem of disparity. In *Proceedings of the Eleventh International Conference on Computational Linguistics*, Los Altos, CA: Morgan Kaufmann Publishers, 1986b.

David Ngi Chin. *Intelligent Agents as a Basis for Natural Language Interfaces*. Doctoral dissertation. Technical Report UCB/CSD 88/396, Computer Science Division, University of California, Berkeley, 1988.

Charles A. Cox. ALANA: Augmentable LANguage Analyzer. Master's thesis. Technical Report UCB/CSD 86/283, Computer Science Division, University of California, Berkeley, 1986.

Jon Doyle. A truth maintenance system. *Artificial Intelligence* 12(3):231-272, 1979.

Gilles Fauconnier. *Mental Spaces: Aspects of Meaning Construction in Natural Language*. The MIT Press, 1985.

Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189-208, 1971.

Barbara J. Grosz. The representation and use of focus in a system for understanding dialogs. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann Publishers, 1977.

Barbara J. Grosz. Focusing in dialog. In *Proceedings of TINLAP-2*. Urbana-Champagne, IL, 1978.

Barbara Grosz and Candace L. Sidner. The structures of discourse structure. Technical Report CSLI-85-39, Center for the Study of Language and Information. 1985.

Matthias Hecking. How to use plan recognition to improve the abilities of the intelligent help system SINIX Consultant. Technical Report 21, Department of Computer Science, University of Saarbruecken, W. Germany, 1987.

Matthias Hecking, Christel Kemke, Erich Nessen, Dietmar Dengler, Michael Gutmann and Günter Hector. The SINIX Consultant -- A progress report. Memo 28, Department of Computer Science, University of Saarbruecken, W. Germany, 1988.

James Hendler. *Integrating Marker-Passing and Problem Solving: A Spreading Activation Approach to Improved Choice in Planning*. Doctoral Dissertation. Technical Report TR-1624, Department of Computer Science, University of Maryland, College Park, 1986. Also published as Technical Report TR-86-01, Brown University, 1986.

Julia Hirschberg. Toward a redefinition of yes/no questions. In *Proceedings of the Tenth International Conference on Computational Linguistics*, 1984.

Christel Kemke. The SINIX Consultant: Requirements, design, and implementation of an intelligent help system for a UNIX derivative. Technical Report 11, Department of Computer Science, University of Saarbruecken, W. Germany, 1986.

Alfred Kobsa and Wolfgang Wahlster, eds. *The Relationship Between User Models and Discourse Models: Two Position Papers*. Technical Report 27, Department of Computer Science, University of Saarbruecken, W. Germany, 1987.

Wendy G. Lehnert. A conceptual theory of question answering. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann Publishers, 1977.

Stephen C. Levinson. *Pragmatics*. Cambridge University Press, 1983.

Diane J. Litman. *Plan Recognition and Discourse Analysis: An Integrated Approach for Understanding Dialogues*. Doctoral dissertation. Technical Report TR170, University of Rochester, 1985.

Diane J. Litman and James F. Allen. A plan recognition model for clarification subdialogues. In *Proceedings of the Tenth International Conference on Computational Linguistics*, 1984.

Marc Luria. Goal conflict concerns. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann Publishers, 1987.

Marc Luria. *Knowledge Intensive Planning*. Doctoral dissertation. Technical Report UCB/CSD 88/433, Computer Science Division, University of California, Berkeley, 1988.

Steven L. Lytinen. Are vague words ambiguous? In Small et al., 1988.

William C. Mann, James A. Moore, and James A. Levin. A comprehension model for human dialogue. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann Publishers, 1977.

Erich Nessen. SC-UM: User modeling in the SINIX Consultant. Memo 18, Department of Computer Science, University of Saarbruecken, W. Germany, 1987.

Peter Norvig. Six problems for story understanders. In *Proceedings of the National Conference on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann Publishers, 1983.

Peter Norvig. *A Unified Theory of Inference for Text Understanding*. Doctoral dissertation. Technical Report 87/339, Computer Science Division, University of California, Berkeley, 1987.

Martha E. Pollack. Good answers to bad questions: Goal inference in expert advice-giving. Technical Report MS-CIS-84-15, University of Pennsylvania, 1984.

Alex Quilici. Detecting and responding to plan-oriented misconceptions. In Wolfgang Wahlster and Alfred Kobsa, Eds., *User Models*. Springer-Verlag, To appear.

Alexander E. Quilici, Michael G. Dyer, and Margot Flowers. Understanding and advice giving in AQUA. Technical Report UCLA-AI-85-19, Computer Science Department, University of California, Los Angeles, 1985

Ashwin Ram. AQUA: Asking Questions and Understanding Answers. In *Proceedings of the National Conference on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann Publishers, 1987.

Elaine Rich, Jim Barnett, Kent Wittenburg, and David Wroblewski. Ambiguity procrastination. In *Proceedings of the National Conference on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann Publishers, 1987.

Roger C. Schank. Interestingness: Controlling inferences. *Artificial Intelligence* 12(3):273-297, 1979.

Roger C. Schank. *Explanation Patterns: Understanding Mechanically and Creatively*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1986.

Roger C. Schank and Robert P. Abelson. *Scripts Plans Goals and Understanding*. Hillsdale NJ: Lawrence Erlbaum Associates, 1977.

C. F. Schmidt, N. S. Sridharan, and J. L. Goodson. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence* 11(1,2):45-83, 1978.

James G. Schmolze and Thomas A. Lipkis. Classification in the KL-ONE knowledge representation system. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Los Altos, CA: Morgan Kaufmann Publishers, 1983.

Ethel Schuster and Timothy W. Finin. Understanding misunderstandings. Technical Report MS-CIS-83-12, Department of Computer and Information Science, University of Pennsylvania, 1983.

John R. Searle. *Speech Acts*. Cambridge University Press, 1969.

Candace L. Sidner. Plan parsing for intended response recognition in discourse. *Computational Intelligence* 1(1):1-10, 1985.

Steven L. Small, Garrison W. Cottrell, and Michael K. Tanenhaus, eds. *Lexical Ambiguity Resolution*. San Mateo, CA: Morgan Kaufmann Publishers, 1988.

Larry Travis and Jeffrey A. Jackson. Automatic identification and naming of complex objects implicit in a relational database. Technical Report CS-TR-2005, Computer Science Department, University of Maryland at College Park, 1988. Also published as Technical Report UMIACS-TR-88-23, University of Maryland Institute for Advanced Computer Studies, 1988.

Wolfgang Wahlster and Alfred Kobsa. Dialogue-based user models. In *Proceedings of the IEEE* 74(7):948-960, 1986.

Robert Wilensky. *Understanding Goal-Based Stories*. Doctoral dissertation. Research Report 140, Computer Science Department, Yale University, 1978.

Robert Wilensky. *Planning and Understanding: A Computational Approach to Human Reasoning*. Reading, MA: Addison-Wesley 1983.

Robert Wilensky. Some problems and proposals for knowledge representation Technical Report 87/351, Computer Science Division, University of California, Berkeley, 1987a.

Robert Wilensky. Primal content and actual content: An antidote to literal meaning. Technical Report 87/365, Computer Science Division, University of California, Berkeley, 1987b.

Robert Wilensky, James Mayfield, Anthony Albert, David Chin, Charles Cox, Marc Luria, James Martin, and Dekai Wu. UC - A progress report. Technical Report 87/303, Computer Science Division, University of California, Berkeley, 1986.

Robert Wilensky, David Chin, Marc Luria, James Martin, James Mayfield, and Dekai Wu. The Berkeley Unix Consultant project. To appear in *Computational Linguistics*, 1989.