RECOGNIZING AND RESPONDING TO PLAN-ORIENTED MISCONCEPTIONS

Alex Quilici
Michael Dyer
Margot Flowers

January 1988
CSD-880002

Recognizing and Responding to Plan-Oriented Misconceptions

Alex Quilici
Michael G. Dyer
Margot Flowers

January 1988

Technical Report UCLA-AI-88-1

Recognizing and Responding to Plan-Oriented Misconceptions

Alex Quilici
Michael G. Dyer
Margot Flowers

Abstract

This paper * discusses the problem of recognizing and responding to plan-oriented misconceptions in advice-seeking dialogues, concentrating on the problems of novice computer users. A cooperative response is one which not only corrects the user's mistaken belief but also addresses the missing or mistaken user beliefs that led to it. Responding appropriately to a potentially incorrect user belief is presented as a process of (1) checking whether the advisor holds the user's belief, (2) confirming the belief as a misconception by finding an explanation for why the advisor does not hold this belief, (3) detecting the mistaken beliefs underlying the misconception by trying to explain why the user holds the incorrect belief, and (4) providing these explanations to the user. An explanation is shown to correspond to a set of advisor beliefs, and searching for an explanation to proving whether various abstract configurations of advisor beliefs hold. A taxonomy of domain-independent explanations for potential user misconceptions involving plan applicability conditions, preconditions, and effects is presented.

*The work reported here was supported in part by a grant from the Lockheed Software Technology Center (Austin, Texas). This paper appears in the Journal of the Association for Computational Linguistics special issue on user modeling in natural language dialog systems (1988). An extended and revised version of this paper by the first author appears in User Modeling in Dialog Systems (Springer Verlag, 1988). Special thanks go to Mike Gasser, John Reeves, and Ron Sumida for fighting their way through several earlier incoherent and uninteresting versions of this paper. Comments by several anonymous reviewers have also greatly improved its organization and content.

1 Introduction

A novice computer user having trouble performing a task often describes his problem to an expert, who is then expected to explain its cause and provide its solution. Consider, for example, the following advice-seeking dialogue between a novice UNIX user and his more experienced counterpart.

User: I tried to remove a file with the "rm" command. But the file was not removed and the error message was permission denied. I checked and I own the file. What's wrong?

Advisor: To remove a file, you need to be able to write into the directory containing it. You do not need to own the file.

User: How do I make that directory writeable?

Advisor: To make a directory writeable, use the "chmod +w" command.

User: Wait a second while I try it. No, "chmod +w" prints the error message "permission denied".

Advisor: To make a directory writeable, you need to own it.

User: So to remove a file, I have to own the directory that contains it.

Advisor: No, to remove a file, you need to have write permission on the directory that contains it. You do not need to own the directory that contains it. You need to own that directory when you do not already have write permission on it.

User: So how do I remove the file?

Advisor: Send mail to whoever has write permission on the directory, asking him to remove the file for you.

Participating as the advisor in such a dialogue requires the ability to recognize and respond to missing or mistaken user beliefs about plan applicability conditions, preconditions, and effects. The advisor above recognizes two user misconceptions. The user first incorrectly believes that owning a file is a precondition to removing it, and then incorrectly believes that the precondition is owning the directory containing it. This advisor also notices several gaps in the user's knowledge. The user has no plan for making a directory writeable, does not know why the advisor's plan for doing so failed, and has no plan for removing a file when the directory in which the file resides is not writeable.

A cooperative advisor response to a *missing user belief* simply provides that belief. The advisor above provides the missing beliefs that the plan for making a directory writeable is to use the "chmod +w" command, that owning a directory is a precondition to making it writeable, and that the plan for removing a file when all else fails is to send mail requesting its removal to some person who has write permission on the directory in which it resides.

However, for a *mistaken belief* the advisor must not only point out that the belief is incorrect and provide a correction, but must also address the missing or mistaken beliefs that are the source of this misconception. Above, this is done by pointing out that the actual precondition is being able to write into the directory that contains it, and by explaining that owning that directory is necessary only if it is not already writeable.

In this paper we examine the problem of detecting and responding to plan-oriented user misconceptions. This problem can be broken into several subproblems:

1. Mapping the user's natural language problem description into a set of user beliefs.

2. Determining which of these beliefs are incorrect.

3. Inferring the missing or mistaken user beliefs that might have led to these incorrect beliefs.

4. Selecting the advisor beliefs to present to the user as a conversationally-cooperative response.

5. Mapping these advisor beliefs into a natural language response.

Here we provide a computational model of (2), (3), and (4). The input is a set of potentially incorrect user beliefs. The output is a set of advisor beliefs to present to the user which correct any mistaken user beliefs and address the missing or mistaken user beliefs that may have led to them. We consider three types of user beliefs: those involving plan applicability conditions (whether a particular plan should be used to achieve a goal), enablements (whether a particular state must exist before a plan can achieve a goal), and effects (whether a state will exist as a result of a plan's execution).

2 An Explanation-Based Approach

How can an advisor determine whether a particular user belief is mistaken and understand how the user came to believe it? And furthermore, how can the advisor determine the contents of a cooperative response to the user's misconception?

An advisor presented with a user belief must do several things. He must first determine whether he shares the user's belief. If he does, it is clearly not a misconception. Assuming that he does not share that belief, the advisor must confirm that it is, in fact, a misconception, and then decide which user beliefs led to it. (If the advisor cannot confirm that the user's belief is mistaken, it could become a new advisor belief.)

We suggest an explanation-based approach to accomplish these tasks. To confirm that the user's belief is a misconception, the advisor tries to find an explanation for why he does *not* hold the user's belief. To infer the problematic user beliefs underlying the user's mistaken belief, the advisor tries to find an explanation for why the user *does* hold this belief. These two explanations constitute the advisor's response to the user.

We illustrate our approach by showing how the advisor arrives at the response found in this exchange from our introductory dialogue.

> **User:** So to remove a file, I have to own the directory that contains it.
>
> **Advisor:** No, to remove a file, you need to have write permission on the directory that contains it. You do not need to own the directory that contains it. You need to own that directory when you do not already have write permission on it.

Here the user's belief is that a file cannot be removed without owning the directory in which it resides.

The advisor first tries to verify that he holds the user's belief. In this case he cannot, so he must now try to determine the reason why he does not hold this belief. The explanation the advisor finds is that the user's belief is contradicted by his belief that a file can be removed without owning the directory in which it resides, and that the user's belief can be replaced by his belief that a file cannot be removed without write permission on that directory.

At this point the advisor has confirmed that the user's belief is a misconception. Now he must try to discover which user beliefs led to this error. To do so, the advisor tries to understand why the user holds this erroneous belief. His explanation is that the user is unaware that to remove a file it is

really only necessary to have write permission on the directory containing it, and that owning that directory is necessary only if one does not already have write permission on it. In other words, the user is unaware that owning a directory is not a precondition for removing a file, but a precondition for achieving one of its preconditions.

Once the advisor finds these explanations, he presents them to the user as the response to his misconception. The response corrects the user's misconception by pointing out that the user's claimed precondition for removing a file is incorrect, by providing the actual precondition for removing a file, and by providing the missing user beliefs that led to the user's misconception.

2.1 Other Work in Explanation-Based Understanding

Our approach derives from work in explanation-based story understanding [37,13,42,43,10,36]. The basic idea is that to understand a particular input, such as a person's action, we have to explain why it has occurred. One way to find an explanation for a person's action is to relate it to known goals the person is trying to achieve. Suppose, for example, that a story understander reads that a hungry character bought a restaurant guidebook [42]. One explanation for this action is that hungry people want to eat, to eat you have to be near food, to be near food you have know where it is and then go there. The guidebook says where the food is. This explanation can be constructed either by using rules to build a reasoning chain or by applying preexisting schemas that capture the relationship.

Our task may be thought of as trying to understand why an actor (either the user or advisor) does or does not hold a particular belief, a task similar to that faced by explanation-based story understanders. Because of the similarities in our tasks, we use the same approach, trying to construct a potential explanation for the beliefs we are trying to understand.

2.2 The Rest of the Paper

Subsequent sections of the paper present our approach in more detail. First, we describe the representation we use to represent plan-oriented user and advisor beliefs. Then, we examine the process by which the necessary explanations are found and provide a taxonomy of explanations for the types of beliefs we consider. Finally, we show how our approach compares with other work in detecting and correcting user misconceptions.

3 Representing User and Advisor Beliefs

The mistaken user beliefs that we consider involve plan applicability conditions, enablements, and effects. In this section we describe how these beliefs are represented. In essence, we make use of existing frameworks for representing planning knowledge, except that we are careful to distinguish between user and advisor beliefs.

Traditional planning systems [15,34] represent an agent's planning knowledge as a database of operators associated with applicability conditions, preconditions, and effects. Since these systems have only one agent, the planner, the entries in the database are implicitly assumed to represent that agent's beliefs. However, because user misconceptions occur when the user's planning knowledge differs from the advisor's, systems that deal with user misconceptions must explicitly distinguish between advisor beliefs about what the user knows and advisor beliefs about what the advisor knows.

Our representation for beliefs [1,2] is similar to that used by existing systems that keep track of the possibly contradictory knowledge of multiple participants [3,4,16,29]. A **belief** relation represents an advisor's belief that an actor maintains that a particular plan applicability condition, precondition, or effect holds. The actor is either the user or the advisor.

belief(user, R)	Advisor believes that user maintains *R*
belief(advisor, R)	Advisor believes that advisor maintains *R*

In this paper we do not discuss beliefs involving other relationships, such as a belief that an object has a particular property. In addition, for readability we do not use the **belief** predicate here but instead precede a list of planning relationships with either "the user believes " or "the advisor believes".

3.1 Representing Planning Relationships

The planning relation can be one of the relations between actions and states shown below. Here *A* denotes an action, which is either a primitive operator whose execution results in a set of state changes, or a plan, which is a sequence of these operators. *S*, *S1*, and *S2* denote states, which are descriptions of properties of objects.

causes(A,S)	Executing A has S an effect
!**causes**(A,S)	Executing A does not have effect S
enables(S1,A,S2)	S1 is necessary for A to have S2 as an effect
!**enables**(S1,A,S2)	S1 is unnecessary for A to have S2 as an effect

applies(A,S)	A is a correct or normal plan for achieving goal state S
!**applies**(A,S)	A is not a plan for achieving S
precludes(S1,S2)	S1 and S2 cannot exist simultaneously
!**precludes**(S1,S2)	S1 and S2 can exist simultaneously
goal(A,S)	Actor A wants to achieve S

These relationships are derived from existing representations. SPIRIT's [29] representation for planning knowledge uses **gen** to represent a state resulting in an action and **cgen** to represent a state resulting in an action only if some other state exists. **causes** and **enables** are identical in semantics to **gen** and **cgen**. **applies**, which has no analog in SPIRIT, is similar to the **intends** relation in BORIS [13]. The difference between **causes** and **applies** is in whether the action is *intended* to cause the state that results from its execution to exist. **causes** represents cause-effect relations which are nonintentional while **applies** represents a cause-effect relation between an action (sequence) or plan which is intended to achieve a desired state (a goal). An action **causes** a state whenever the state results from its execution. An action **applies** to a state when an actor believes the action will cause the desired state to occur.

To see why this distinction is necessary, consider two actions that can be used by a user who wants to remove one of his files: typing "rm" followed by the file's name, and typing "rm *". Both have removing the file as one of their effects, but the latter also causes all other files to be removed as well, an effect that is *not* the user's *goal*. Only "rm file" **applies** to removing a file, although both actions have an effect that **causes** the file to be removed.

To further illustrate the semantics of these relations, we show how they can be used to represent the first exchange in our example dialogue.

> **User:** I tried to remove a file with the "rm" command. But the file was not removed and the error message was permission denied. I checked and I own the file. What's wrong?

> **Advisor:** To remove a file, you need to be able to write into the directory containing it. You do not need to own the file.

Three of the user's beliefs in this exchange are: (1) the "rm" command is used when one wants to remove a file, (2) one has to own a file to remove it, and (3) an error message resulted when the plan was executed. In terms of these planning relations, the user's beliefs are:

> **applies**(using "rm file", the file's removal)
> **enables**(owning the file, using "rm file", the file's removal)
> **causes**(using "rm" on the user's file, an error message)

The advisor holds several similar beliefs, except that he believes that to remove a file it is necessary to have write permission on the directory containing it. In terms of the planning relationships, the advisor's beliefs are:

applies(using "rm file", the file's removal)
enables(directory write permission, using "rm", the file's removal)
causes(using "rm" on the user's file, an error message)

(The paper is not concerned with representing notions such as "the file's removal" or "write permission on the directory containing the file". The details of the representation for such things may be found in [32].)

The user and advisor in this exchange share one belief that we have not represented. This belief is that using "rm" did not cause the user's file to be removed. To represent beliefs that a state did not result from an action, that a plan is not applicable to a goal, or that a state is not an enablement condition of an action having another state as a result, we use **!causes**, **!applies**, and **!enables**, respectively. The belief above is represented with **!causes**, a belief that "mkdir" is not used to remove a file is represented with **!applies**, and a belief that "rm" does not require owning the directory containing the file is represented with **!enables**.

!causes(using "rm" on the user's file, the file's removal)
!applies(using "mkdir file", the file's removal)
!enables(owning the directory, using "rm", the file's removal)

It is also necessary to be able to represent the notion that a state's existence caused a planning failure. Consider the following exchange.

User: I accidentally hit the up arrow key and it deleted 20 unanswered mail messages. How can I get them back?

Advisor: Hitting the up arrow does not delete your messages, but does result in your being disconnected from the etherplexer. You could not access your mail messages because they were moved to "mbox". The mail program requires that your mail messages be in "mailbox".

Here the advisor believes that the user's mail messages are inaccessible because they are not in the location the mail program expects them to be. The belief that the mail program requires the mail messages to be in the file "mailbox" can be represented using **enables**. The advisor's belief that the mail messages being in the file "mbox" prevents the mail program from accessing is represented with **precludes**, which captures the notion that two states are mutually exclusive.

enables(messages in "mailbox", use "mail", display messages)
precludes(messages in "mbox", messages in "mailbox")

precludes and **!precludes** relations between states can be inferred using rules such as "an object cannot be in two places at once".

The one other relation we find useful is **goal**, which is used in representing a belief that an actor wants to achieve a particular state. In the example above, the advisor believes that one goal of the user is accessing his mail messages. The advisor's belief is:

goal(user, access user's mail messages)

Most user-modeling systems use a similar relation to explicitly represent that a state is a user's goal.

3.2 Summary of the Representation

The main focus of our work is in trying to detect and respond to user misconceptions. To do so, it is necessary to have some representation for user and advisor planning knowledge. Our representation is based on that used by traditional planning systems. The most important difference is that we take care to distinguish between things the advisor believes and things the advisor thinks the user believes. We also distinguish between actions that are intended to achieve a state and actions that happen to have a particular state as one of their effects. And we find it necessary to represent beliefs that two states cannot exist at the same time and that achieving a particular state is a goal of the user.

4 Explanation-Based Misconception Recognition and Response

Our approach to recognizing and responding to a potentially incorrect user belief revolves around the advisor trying to do several things. First, the advisor tries to verify that he does not share the user's belief. Next, the advisor tries to confirm that the user's belief is a misconception. The advisor does so by finding an explanation for why he does not share the user's belief. After the belief is confirmed as a misconception, the advisor tries to detect its source. He does this by finding a potential explanation for why the user holds that incorrect belief, based on a taxonomy of abstract explanation

classes. Finally, the advisor presents these explanations to the user as a cooperative response.

But what exactly is an explanation? And what knowledge does the advisor need to find one? And, finally, how is an explanation found?

4.1 Explanations as Sets of Beliefs

An explanation is a set of advisor beliefs that accounts for why a particular belief is or is not held. An advisor, presented with a potentially incorrect user belief, has to find two explanations.

The first explanation confirms that the user's belief *is* a misconception. To find this explanation the advisor tries to find a set of advisor beliefs that justify his not holding the user's belief. For instance, the user in our earlier example had an incorrect belief that owning a directory is a precondition for removing a file.

> **enables**(own directory, use "rm file", the file's removal)

Two advisor beliefs constitute an explanation for why the advisor does not hold this belief. The first is the advisor's contradictory belief that owning a directory is not a precondition for removing a file. The other is his belief that the actual precondition is write permission on the directory containing the file.

> !**enables**(own directory, use "rm file", the file's removal)
> **enables**(writeable directory, use "rm file", the file's removal)

These two beliefs confirmed that the user's belief was mistaken.

The other explanation explains *why* the user holds this incorrect belief. To find this explanation the advisor tries to find a set of advisor beliefs that capture the source of the user's misconception. Two advisor beliefs provide a possible explanation for the incorrect user belief above. The first is that one has to own a directory to make it writeable. The other is that having a writeable directory is the precondition to removing a file.

> **enables**(own directory, use "chmod", obtain writeable directory)
> **enables**(writeable directory, use "rm", the file's removal)

The user's not sharing these advisor beliefs explains the user's misconception, which is that the user does not realize that owning a directory is merely a precondition to obtaining write permission on the directory, which is the actual precondition to removing the file.

4.2 Required Advisor Knowledge

To find an explanation the advisor must have three types of knowledge: (1) a set of domain-specific beliefs, (2) a set of rules for inferring additional beliefs, and (3) a set of abstract explanation patterns. All of these must come from past advisor experience or past advisor interaction with users. However, here we simply assume their existence, and leave understanding how they are obtained for future research.

The first type of required knowledge is a set of domain-specific beliefs about plan applicability conditions, preconditions, and effects. Examples of these include beliefs that "rm" is used to remove a file, and that it is necessary to have write permission on the directory containing the file. Without these types of beliefs it would be impossible for the advisor to correct user misconceptions about the preconditions for removing a file. This category of knowledge includes beliefs such as a belief that "rm" is not used to remove a directory. These negated beliefs — `!applies`, `!enables`, `!causes`, and so on — are especially useful in detecting misconceptions. An advisor, with the explicit belief that "rm" is not applicable to removing a directory, can trivially detect that a user belief that "rm" is applicable to removing a directory is incorrect.

These domain-specific beliefs are assumed to derive from past advisor experiences. An advisor who successfully uses "rm" to remove a file will believe that using "rm" is applicable to the goal of removing a file. An advisor who uses "rm" to try to remove a directory and has it fail will believe that "rm" is not applicable to removing a directory. The negated beliefs correspond to the bug lists kept by many tutoring and planning systems [5,6,7,38].

The second type of advisor knowledge is a set of rules that help infer negated domain-specific beliefs, such as a belief that a particular action does not result in a particular state, or that a given plan is not useful for a particular goal. These rules are needed because the advisor cannot be expected to have a complete set of these beliefs. One such rule, for example, suggests that "if a state S is not among the known states that result from an action A's execution, assume that A is not applicable to achieving S". There are similar rules for the other types of beliefs.

The third and final type of knowledge is a taxonomy of potential explanations for why an actor might or might not hold a belief. Each type of planning relation — `applies`, `enables`, and `effects` — is associated with two sets of potential explanations. One set provides reasons why an actor

might hold a particular belief involving that planning relation. The other set provides reasons why an actor might not.

The inference rules and potential explanations differ for each type of planning relation. Associated each type of planning relation is:

1. a set of rules for inferring its negation (which prove useful in finding explanations for why the belief is or is not held),
2. a potential explanation for why an actor does not hold a belief involving that planning relationship, and
3. a set of potential explanations for why an actor does hold a belief involving that planning relationship.

For example, **applies** is associated with a set of rules for inferring that an actor holds a particular **!applies** belief, a potential explanation for why an actor does not hold a given **applies** belief, and a set of potential explanations for why an actor does hold a given **applies** belief.

5 Potential Explanations

The advisor must be able to find a reason for why a particular belief is or is not held. One way to do so is (1) classify the belief, and (2) try to verify one of the potential explanations associated with that class of belief. A potential explanation is an abstract pattern of planning relationships. The idea is that to verify a potential explanation, the advisor tries to prove, either by memory search or by deductive reasoning, that each of these planning relationships hold.

There are two types of potential explanations. The first explains why an actor does not hold a belief. The other explains why an actor does. In this section we describe the potential explanations associated with the planning relationships we have examined. The following section discusses in detail how they are used.

5.1 Potential Explanations for Not Holding a Belief

The potential explanations for why the advisor *does not* hold an instance of one the plan-oriented beliefs are shown below. Each of these potential explanations suggests that to confirm that a user's belief is a misconception, the advisor must try to verify that one of his beliefs contradicts the user's

belief, and that one of his beliefs can replace it. The only difference between the potential explanations is in the type of belief being contradicted or replaced.

Unshared User Belief	**Potential Explanation**	**English Description**
applies(Ap, Sg)	!applies(Ap,Sg)	Plan is not used to achieve Goal
	applies(A,Sg)	Other plan is used to achieve Goal
enables(Sp,Ap,Sg)	!enables(Sp,Ap,Sg)	State is not precondition of Action
	enables(S,Ap,Sg))	Other state is precondition of Action
causes(Ap,Sp)	!causes(Ap,Sp)	Action does not cause state
	causes(A,Sp)	Other action does cause state

Consider our earlier example in which the user's belief is that a precondition of removing a file is owning the directory containing it. The potential explanation suggests trying to prove that the advisor holds two beliefs: that owning a directory is not a precondition of removing a file, and that some other state is. Here, the advisor finds that he believes that owning a directory is not a precondition of removing a file (either by finding that relationship in his knowledge base or by deducing it). The advisor also finds that directory write permission is a precondition of removing a file. These beliefs explain why the advisor does not hold the user's belief, confirming it as a misconception.

A similar process is used to confirm that the advisor does not hold a user's **applies** or **causes** belief. Consider the following exchange.

> **User:** I tried to display my file with the "ls" command but it just printed the file's name.
>
> **Advisor:** The "ls" command is not used to display the contents of files, the "more" command is. "ls" is used to list the names of your files.

The user's potentially incorrect belief is that using "ls" is applicable to achieving the goal of displaying a file's contents. The potential explanation for why an advisor does not hold this belief is that the advisor does not believe that using "ls" is applicable to this user's goal, and that using "ls" is applicable to some other goal. So the advisor tries to verify (again, by either search or deduction) that "ls" is not applicable to displaying the file's

contents, and he tries to verify that some other plan does. Here the advisor finds that "more" is used instead.

Finally, consider the following exchange.

User: I deleted a file by typing "remove".

Advisor: No, typing remove did not delete your file. Typing "rm" deleted it. Typing "remove" cleans up your old mail messages.

The user's potentially mistaken belief is that typing remove results in a file being deleted. The potential explanation for why the advisor does not share this belief is that the advisor instead believes that typing "remove" does not result in a file being deleted and that some other action does. The advisor verifies that typing "remove" does not cause a file to be deleted and that "rm" is an action that does.

5.2 Explanations for Holding a Belief

The potential explanations we have examined so far explain why an actor does not hold a particular belief. There are also potential explanations for why an actor *does* hold an incorrect belief. We now present a taxonomy of these explanations for each of the three types of beliefs.

5.2.1 Explanations for Incorrect Applies

There are four potential explanations for why a user holds an incorrect **applies** belief of the form **applies**(*Ap*, *Sp*). Recall that to recognize that this type of user belief is incorrect the advisor found two beliefs of the form **!applies**(*Ap*, *Sp*) and **applies**(*A*, *Sp*). Here are the potential explanations along with English descriptions for each.

Class Of Mistake	**Potential Explanation**	**English Description**
Plan Achieves Different Goal	!causes(Ap,Sp) applies(Ap,S)	No effect achieves goal Plan applies to other goal
Plan Missing Effect	!causes(Ap,S) causes(A,S)	User plan does not an effect that other plan has
Unachievable Plan Enablement	enables(S,Ap,Sp) !causes(A,Sp)	Some state enables Plan No action achieves this state

Plan Thwarts User Goal	causes(Ap,S)	User plan has effect
	precludes(S, Sp)	that thwarts the user's goal
	goal(user, Sp)	
	!causes(Ap,S)	Advisor plan does not have that effect

The first, **Plan Achieves Different Goal**, explains one of our earlier examples. The explanation is that the user is unaware that his plan does not have an effect that achieves his goal, and that his plan is, in fact, used to achieve some other goal.

User: I tried to display my file with the "ls" command but it just printed the file's name.

Advisor: The "ls" command is not used to display the contents of files, the "more" command is. "ls" is used to list the names of your files.

The user's incorrect belief that using "ls" displays a file arises because the user is unaware of two things. The first is that using "ls" does not display the contents of files; the other is that "ls" is applicable to listing the names of files.

The second, **Plan Missing Effect**, suggests that the user is unaware that his plan P1 does not have one of the effects that the plan P2 (that achieves his goal) has.

User: I tried to remove my directory and I got an error message "directory not empty". But "ls" didn't list any files.

Advisor: Use "ls -a" to list all of your files. "ls" cannot be used to list all of your files because "ls" does not list those files whose names begin with a period.

The user's mistaken belief is that "ls" should be used to list all file names. This belief arises because the user is unaware that "ls" does not have an effect that causes it to list files whose names begin with a period, an effect that the correct plan (ls -a) has.

The third, **Unachievable Plan Enablement**, suggests that the user is unaware his plan will not work because there is no plan to achieve one of its enablements.

User: So to read Margot's mail, all I have to do is "more ~flowers/mail".

Advisor: No, only "flowers" can read her mail.

The user mistakenly believes that his plan of using "more" to examine Margot's mail file will allow him to read her mail. The advisor believes that "more" has an effect of displaying a user's mail, that one of its enablements is that you have to be that particular user, and that no plan has an effect that achieves this enablement.

The last, **Plan Thwarts User Goal**, suggests that the user is unaware that another plan achieves the user's goal without an additional effect that the user's plan has.

> **User:** To list files whose names begin with a number, I pipe "ls" to "grep [0-9]".
>
> **Advisor:** Use "ls [0-9]*" instead. It is more efficient.

The user's mistaken belief is that piping "ls" to "grep" is the most appropriate plan for listing files whose names begin with a digit. The user's misconception arises because he is unaware that the plan of using "ls[0-9]*" not only achieves his goal but also does not thwart his other goal of using his time efficiently.

5.2.2 Explanations for an Incorrect Enables

Just as there are several different sources of user misconceptions about a plan's applicability to a goal, there are also several different sources of user misconceptions about whether a state is a precondition to a plan achieving a goal; that is, a user belief of the form **enables**(*Se*, *Ap*, *Sp*). Recall that to recognize that this type of belief is incorrect the advisor found two beliefs of the form **!enables**(*Se*, *Ap*, *Sp*) and **enables**(*S*, *Ap*, *Sp*). Here are the potential explanations along with English descriptions for each.

Class Of Mistake	**Potential Explanation**	**English Description**
Enablement For Subgoal	enables(Se,A,S)	State enables actual enablement
Enablement For Only One Plan	causes(A,Sp) !enables(Se,A,Sp)	Plan achieves user's goal without claimed enablement
Enablement Too Specific	causes(A1,Se) causes(A1,S) causes(A2,S) !causes(A2,Se)	User enablement results from action that achieves real enablement Other action achieves real enablement without user's enablement as effect

The first, **Enablement For Subgoal**, explains the user's mistake in our introductory exchange. The explanation is that the user is unaware that his precondition is not a precondition of the goal itself, but of one of its preconditions.

User: So to remove a file, I have to own the directory that contains it.

Advisor: No, to remove a file, you need to have write permission on the directory that contains it. You do not need to own the directory that contains it. You need to own that directory when you do not already have write permission on it.

This user is unaware that owning a directory is a precondition for achieving write permission on it, and that having write permission is a precondition for removing a file.

The second, **Enablement for One Plan**, suggests that the user is unaware that a plan without his claimed precondition achieves his goal.

User: So I can only edit files when I'm on a smart terminal?

Advisor: Only if you edit with "re". "vi" works fine on a dumb terminal.

The user's incorrect belief is that it is necessary to have a smart terminal to edit a file. This belief arises because the user is unaware that only one plan, using "vi", requires a smart terminal, and that there are other plans that do not.

The last, **Enablement Too Specific**, suggests that the user is unaware that his precondition is less general than the actual precondition for achieving his goal.

User: So I have to remove a file to create a file?

Advisor: You do not have to remove a file to create a file. You must have enough free space. Removing a file is only one way to obtain it. You could also ask the system administrator for more space.

The user mistakenly believes that it is necessary to remove an existing file before a new file can be created. The advisor believes that the precondition is sufficient space for the new file, which can be achieved either by executing a plan for removing a file or by executing the plan of requesting more space.

5.2.3 Explanations for Incorrect Causes

One final class of user misconception is an incorrect belief that a particular state results from a plan's execution; that is, a user belief of the form **causes**(*Ap*, *Sp*). Recall that to recognize that this type of belief is incorrect the advisor found beliefs of the form **!causes**(*Ap*, *Sp*) and **causes**(*A*, *Sp*). There are three potential explanations for this type of mistaken belief.

Class Of Mistake	**Potential Explanation**	**English Description**
Plan has Other Effect	applies(Ap,So)	Action used to cause other effect
Effect Requires Enablement	enables(S,Ap,Sp) !causes(A,S)	State required for Action and no way to achieve State
Effect Inferred From Other Effect	causes(Ap,So) precludes(So, S) precludes(Sp, S)	Action causes other effect That effect precludes a state that is precluded by user's effect

The first, **Effect From Another Plan**, accounts for an earlier example. The explanation is that the user is unaware that the user's action actually has a different effect.

> **User:** I deleted a file by typing "remove".
>
> **Advisor:** No, typing remove did not delete your file. Typing "rm" deleted it. Typing "remove" deletes a mail message from the mail program.

The user's mistaken belief is typing "remove" deletes a file. The user is unaware that typing "remove" actually throws away old mail messages.

The second, **Effect Requires Unfulfilled Enablement**, suggests that the user is unaware that a particular state is required for the plan to have the claimed effect.

> **User:** I was cleaning out my account when I accidentally deleted all the command files by typing "rm".
>
> **Advisor:** You can't delete the command files with "rm" unless you are the system administrator.

The user incorrectly believes that typing "rm" resulted in the removal of various system files. The advisor believes that it is necessary for the user to be the system administrator for this effect to occur.

The last, **Effect Inferred From Other Effect**, accounts for another one of earlier examples. It suggests that the user is unaware that one effect of his plan has incorrectly led him to believe what was another effect of the plan.

> **User:** I accidentally hit the up arrow key and it deleted 20 unanswered mail messages. How can I get them back?
>
> **Advisor:** Hitting the up arrow does not delete your messages, but does result in your being disconnected from the etherplexer. You could not access your mail messages because they were moved to "mbox". The mail program requires that your mail messages be in "mailbox".

The user incorrectly believes that one effect of hitting uparrow was that his mail messages were deleted. This belief occurs because the user is unaware that one effect of hitting uparrow is that files are moved to a different location, which makes them seem inaccessible.

6 A Detailed Process Model

We have presented three sets of potential explanations and briefly sketched how they are used. In this section we provide a more detailed view of the process by which an explanation is found.

An advisor presented with a user belief has three goals. First, he wants to know whether he shares the user's belief, Second, he wants to confirm that the user's belief is indeed a misconception. Third, he wants to infer the reason's behind the user's mistake.

The advisor accomplishes the first by trying to verify that he holds the user's belief. He accomplishes the second by trying to find an explanation for why he does not hold the user's belief. He accomplishes the third by trying to find an explanation for why the user does hold that belief.

Two questions need to be answered. How does the advisor verify that he holds a particular belief? And how does the advisor explain why he does not hold a belief, or why the user does?

6.1 Verifying An Advisor Belief

Verifying whether or not the advisor believes that a particular planning relationship holds takes two steps. First, the advisor searches his memory for the desired piece of planning knowledge. Then, if it is not found, the advisor applies the set of rules associated with that planning relationship to try and prove that it holds. Once the advisor has proved that the planning relationship holds, either by search or by reasoning, that piece of knowledge is noted to be an advisor belief.

Consider, for example, the process of verifying that the advisor holds a belief that owning a directory is not a precondition of removing a file. If this fact is already known from past experience, the advisor will recognize it during memory search. If not, the advisor can try to deduce it. One rule that applies here says that "if a state S is not one of the known states that are preconditions to an action A for achieving a goal state, then assume that S is not a precondition". Here this means that if owning a directory is not among the known preconditions for removing a file, assume it is not a precondition for removing a file.

6.2 Finding An Explanation

The advisor must be able to explain why an actor does or does not hold a particular belief. Finding an explanation is accomplished by hypothesizing one associated with the given class of belief and then trying to confirm it. The advisor:

1. Classifies the belief according to its type: **applies**, **enables**, or **effects**.
2. Selects one of the potential explanations associated with that class of belief. The potential explanation is an abstract configuration of planning relationships.
3. Instantiates this potential explanation with information from the user's belief.
4. Tries to verify each of the planning relationships within the potential explanation. If all can be verified, this potential explanation is the desired explanation.
5. Repeats the process until one of the potential explanations associated with this belief's type is verified or all potential explanations have been tried and have failed.

The result of the process of finding an explanation is that the advisor has verified that he holds a particular set of beliefs. These beliefs constitute the desired explanation.

6.3 An example

This section is a detailed look at the advisor's processing of the user belief that owning a directory is a precondition of removing a file.

> **enables**(user own directory, use "rm", the file's removal)

First, the advisor tries to verify that he holds the user's belief. He cannot.

Next, the advisor tries to confirm that the user's belief is, in fact, a misconception. He does this by trying to explain why he does not hold this user belief. He notes that it can be classified as a belief that some state *Sp* (owning the directory) is a precondition to achieving some other state *Sg* (removing a file). The potential explanation for why the advisor does not hold this type of belief is that he believes that *Sp* is not a precondition of achieving *Sg*, and that some other state *S* is a precondition of *Sg*. By instantiating this potential explanation, the advisor determines that he must check whether he holds beliefs that:

> !**enables**(owning a directory, use "rm file", the file's removal)
> **enables**(*S*, use "rm file", removing a file)

The advisor finds that he believes that owning a directory is not a precondition of removing a file (either by finding that relationship in memory or by deducing it). The advisor also finds that write permission on a directory is a precondition of removing a file (that is, that *S* can be instantiated with write permission on a directory). These matching beliefs confirm that the user's belief is a misconception.

Now, the advisor has to try to find an explanation for why the user holds this mistaken belief. One potential explanation is that the user is unaware that *Sp* is actually a precondition of achieving a state *S*, which is a precondition to achieving *Sg*. In this case, instantiating *Sp* and *Sg* leads to the advisor to try and verify that he holds two beliefs:

> **enables**(*S*, use "rm file", the file's removal)
> **enables**(owning a directory, *A*, *S*)

These beliefs are verified when the advisor finds that having write permission on a directory is a precondition to removing a file, and that owning a directory is a precondition to obtaining write permission the directory. The potential explanation suggests that the user's misconception resulted from his being unaware of these two advisor beliefs.

Finally, the advisor presents the resulting beliefs to the user. The user is informed of the beliefs used to confirm the user's misconception and the beliefs used to explain its source.

6.4 The Point of Potential Explanations

Having a taxonomy of potential explanations lessens the amount of reasoning the advisor must do to detect and respond to the user's misconceptions.

To see why, consider an advisor trying to understand how the user arrived at the mistaken belief that a precondition of removing a file is owning the directory containing it. The advisor is trying to find some connection between the user's enablement and removing a file. The potential explanations suggest how to find specific, likely-to-be-useful connections. For example, the potential explanation **Enablement for Subgoal** suggests examining whether achieving any of the preconditions of removing a file requires owning a directory.

Without a set of potential explanations, it becomes necessary to reason from a set of rules that describe likely differences between user and advisor beliefs. One rule might be that a user may incorrectly attribute an enablement of one action to another action. Another rule might be that a user may incorrectly attribute the result of one action to another action. From a set of such rules the advisor must somehow deduce the cause of the user's mistake. By using potential explanations the problem becomes instead one of guided memory search rather than reasoning from first principles.

7 Related Work

Two approaches have been used to detect and correct misconceptions. The first approach is used by many intelligent tutoring systems [5,6,7,38]. These systems locate mistaken beliefs in a database of error-explanation pairs and provide the associated explanation. A basic problem with this approach is that, because there is no information about the underlying causes of the errors, these systems can handle only those misconceptions known in advance.

The other approach avoids the difficulty inherent in enumerating all possible misconceptions within a domain by using strategies that address an entire class of misconceptions. The user's misconception is classified according to the abstract reasoning error likely to have led to it. This approach shares many features with recognizing abstract thematic situations (such as irony) in narratives, where such situations are defined in terms of abstract planning errors made the narrative characters [13,14,12]. Once an appropriate strategy is found, it can be used to generate advice (in narratives, this advice may be in the form of adages). In advisory systems, this approach has been applied to both object- and plan-oriented misconceptions.

7.1 Object-Oriented Misconceptions

ROMPER [24,25] corrects user misconceptions dealing with whether an object is an instance of a particular class of objects or possesses a particular property.

User: I thought whales were fish.

ROMPER: No, they are mammals. You may have thought they were fish because they are fin-bearing and live in the water. However, they are mammals since, while fish have gills, whales breathe through lungs and feed their young with milk.

ROMPER classifies a user's misconception as either a misclassification or misattribution and then selects one of several strategies associated with each class of misconception to generate a response. Each strategy addresses a different type of reasoning error, and is selected based on ROMPER's own beliefs about objects and its model of the user's relevant beliefs. One such strategy is useful when the advisor believes that *X isa Z*, the user mistakenly believes that *X isa Y*, and the advisor believes that *X* and *Y* share certain attributes. The strategy suggests presenting these shared attributes as a possible reason for the misclassification, and pointing out the unshared attributes that lead the advisor to believe that *X isa Z*.

Despite dealing with a very different class of misconceptions, ROMPER's approach is similar to ours. The major difference is that our explanation-based approach separates the beliefs needed to confirm the user's belief as a misconception from those needed to understand why the user holds it. The strategy above divides into two explanations. The first confirms that a user belief that *X isa Y* is incorrect if the advisor believes that *X isa Z* because *X* and *Z* share certain attributes. The other suggests that the user may hold

this belief because X and Y share certain attributes. The advantage to our approach is that the information regarding the beliefs that confirm that the user has a misconception can be separated from the explanations for why the user holds the belief, and unnecessary duplication of tests is avoided.

7.2 Plan-Oriented Misconceptions

Two efforts have examined detecting and responding to plan-oriented misconceptions.

Joshi, Webber, and Weishedel [21] suggest using a strategy-based approach to provide cooperative responses to problematic planning requests. They consider "How do I do X?" questions in which X can be inferred to be a subgoal of a more important goal Y.

User: How can I drop cs577?

System: It is too late in the quarter to drop it. But you can avoid failing by taking an incomplete and finishing your work next quarter.

They provide several strategies, listed below, for selecting the contents of a reasonable response, with strategy selection based on the advisor's beliefs about which plans achieve a particular goal and the achievability of their preconditions.

	Situation	**Response**
(1)	Unachievable Precondition E of X	Provide E
	(a) Plan P achieves Y	Provide P
	(b) No plan to achieve Y	Point this out
(2)	X doesn't help achieve Y	Point this out
	(a) Plan P achieves Y	Provide P
	(b) No plan to achieve Y	Point this out
(3)	Plan P better way to achieve Y	Provide P
(4)	X only way to achieve Y	Point this out
(5)	Plan P involving uncontrollable event E achieves Y	Provide P

One such strategy, useful when the advisor believes that X cannot be achieved because of an impossible-to-achieve precondition, is to point out the troublesome precondition and suggest an alternate plan that achieves Y.

Our work differs from theirs in several respects. The main difference is that they focus on correcting the user's misconception instead of trying to explain why it occurred. Only one strategy above is concerned with

providing an explanation that addresses the source of a user misconception (in this case, an inappropriate plan). The other strategies describe situations in which achieving X is inappropriate and an alternate plan for Y exists and should be presented to the user as a correction. In addition, they did not consider responding to incorrect beliefs about plan preconditions or effects.

The other effort, SPIRIT [28,29], tries to detect the inappropriate plans underlying queries made by users of a computer mail program and the mistaken user beliefs underlying those plans.

> **User:** I want to prevent Tom from reading my file. How can I set the permissions on it to faculty-read only?
>
> **System:** You can make the file readable by faculty only using "set permission". However, Tom can still read it because he's the system administrator.

User misconceptions about the applicability and executability of plans are detected by reasoning about the likely differences between the advisor's beliefs and the user's, with various rules used to infer these differences. One such rule, used to detect the source of the misconception above, states that an advisor who believes that an act has a particular result under certain conditions can infer that the user has a similar belief missing one of the required conditions.

SPIRIT has task a similar to ours but takes a very different approach, trying to determine the cause of the user's error through reasoning from first principles rather than memory search. In addition, SPIRIT cannot detect or respond to mistakes involving plan applicability conditions or preconditions. Finally, SPIRIT does not specify how knowledge of the cause of the user's mistaken belief affects the information to be included in a cooperative response, something that falls naturally out of our model.

7.3 UNIX Advisors

Finally, there are two other related research efforts, UC [40,41] and SC [20], that address providing advice to novice UNIX users. Neither system, however, detects or responds to misconceptions. Instead, both are concerned with tailoring a response to a question to reflect the user's level of expertise. UC's user modeling component, KNOME [9], analyzes a user's questions to determine which stereotypical class the user belongs to and then uses this information to provide more details and possibly more examples to less experienced users.

Novice: What does the "rwho" command do?

UC: Rwho lists all users on the network, their tty, their login time, and their idle time.

Expert: What does the "rwho" command do?

UC: Rwho is like who, except rwho lists all users on the network.

SC's user modeling component, SCUM [26], takes an approach similar to UC's, also using stereotypical information. These approaches are complementary to ours.

8 Implementation Details

The theory discussed in this paper is embodied in AQUA, a computer program currently under development at UCLA. The current version of AQUA is implemented in T [33], using RHAPSODY [39], a graphical AI tools environment with Prolog-like unification and backtracking capabilities, and runs on an Apollo DN460 workstation. Given a set of user beliefs involving plan applicability conditions, preconditions, or effects, AQUA determines which of these user beliefs are incorrect and what missing or mistaken user beliefs are likely to have led to them, and then produces a set of advisor beliefs that capture the content of the advisor's response. AQUA's domain of expertise is in the basic plans used to manipulate and access files, directories, and electronic mail. It has been used to detect and respond to at least two different incorrect user beliefs in each class of misconception that we have identified. More detailed descriptions of the program's implementation can be found in [31,32].

9 Limitations and Future Work

Our approach to determining why an actor does or does not hold a particular belief has been to let potential explanations direct the search for the advisor beliefs that serve as an appropriate explanation. Our focus has been on discovering and representing these explanations. The limitations of our approach arise in areas we have ignored, each of which is an interesting area of research.

9.1 Inferring the Set of User Beliefs

Our model assumes that the user's problem description has somehow been parsed into a set of beliefs. However, users rarely explicitly state their beliefs, leaving the advisor to the difficult task of inferring them. Consider our introductory exchange.

> **User:** I tried to remove a file with the "rm" command. But the file was not removed and the error message was permission denied. I checked and I own the file. What's wrong?
>
> **Advisor:** To remove a file, you need to be able to write into the directory containing it. You do not need to own the file.

Here the advisor must infer the user's beliefs that (1) using "rm" is applicable to removing a file, that (2) using "rm" did not cause the file's removal, that (3) using "rm" resulted in an error message, and that (4) owning a file is a precondition to removing it.

Inferring the first belief requires a rule such as "if the user tries to achieve a state with a particular action, assume the user believes that action achieves that state". The second belief can be inferred from the rule that "if an utterance describes the nonexistence of a state that is a believed result of an action, assume that the user believes that the action did not cause the state". A similar rule can be used to infer the third belief.

Inferring the final belief, that owning a file is a precondition to its removal, is a difficult task. Because there are a potentially-infinite number of incorrect user beliefs about the preconditions of removing a file, the advisor cannot simply match owning a file against a list of incorrect preconditions. Because the user may have been discussing other plans and other goals the advisor cannot simply assume that any utterance after a plan's failure refers to its preconditions. Instead, the advisor needs to infer this user belief from the knowledge that the user did some sort of verify-action, the knowledge that one plan for dealing with a plan failure is to try to verify that the enablements of the plan have been achieved, and the knowledge that both owning the file and having write permission are different instantiations of having sufficient permission.

Inferring beliefs like these, that involve the user's plans and goals and the relationships between, even when they differ from the advisor's, is currently an active area of research [8,19,17,40,32].

9.2 Retrieving Advisor Beliefs

Our potential explanations suggest patterns of beliefs that the advisor should search for. However, we have not specified how this search of the advisor's memory is actually carried out, how a belief in memory can be retrieved efficiently, or how the beliefs are actually acquired through experience. AQUA's organization of plan-oriented beliefs discussed in [30,32]. It is based on earlier work [22,35] in taking experiences and indexing them appropriately for efficient search and retrieval, especially that involving indexing memory around various planning failures [23,32,18,13].

Because the advisor may need to verify a belief that is not stored directly in memory, memory search may not be sufficient. Suppose the advisor is trying to verify that owning a directory is not required to remove a file. The advisor may be able to deduce this belief from a past experience in which he removed a file from /tmp, a directory owned by the system administrator. Similarly, the advisor may be able to deduce that write permission is needed to remove a file from his beliefs that write permission is needed to make changes on objects and that removing a file involves making a change to a directory. This requires more powerful reasoning capabilities than AQUA's simple rules for inferring negated beliefs.

Finally, AQUA assumes the existence of a taxonomy of planning failures. We have left the automatic creation of this taxonomy from advisor experiences to future research. Initial work in recognizing and indexing abstract configurations of planning relations is discussed in [11,12].

9.3 Other Classes of Misconceptions

We are currently studying how well the classes of misconceptions described here account for responses to misconceptions in domains other than the problems of novice computer users, such as the domain of simple day-to-day planning. In addition, we are examining other classes of planning misconceptions. For example, to respond to an incorrect user belief such as "rm" cannot be used to remove a file, the advisor needs potential explanations for why an action does not apply to a particular goal state.

We do not yet know whether our approach is suitable for generating responses to misconceptions that are not directly related to plan-goal interactions, such as mistakes in referring to an object. Consider the following exchange:

User: Diana is up but I cannot access my file.

Advisor: Your files are on Rhea, not Diana. They moved your files yesterday because your file system was full.

Here the user's problem is that he is incorrectly using "Diana" to refer to the machine his files are on. We are examining whether our approach is extendable to respond to these types of user misconceptions.

9.4 Response Generation

The response we provide is a set of advisor beliefs that is as complete as possible. We make no attempt to use knowledge about other user beliefs to modify our response to provide only the most relevant beliefs. However, if the advisor can infer that a user knows that his plan has failed (perhaps because of the error message a command produces), he need not inform the user that his plan is incorrect. One straightforward way to extend our model is to have the advisor filter out those beliefs he can infer the user has.

The advisor should use information about the user to tailor their response based on the user's level of expertise. Recall the following exchange.

User: I tried to remove my directory and I got an error message "directory not empty". But "ls" didn't list any files.

Advisor: Use "ls -a" to list all of your files. "ls" cannot be used to list all of your files because "ls" does not list those files whose names begin with a period.

An advisor who knows the user is a novice might want to augment his response with an explanation that "-a" is a command option and that command options cause changes in the normal behavior of commands. Several researchers are working on tailoring the response to the user based on knowledge about the user's expertise [9,27].

10 Conclusions

We have presented an explanation-based approach to the problem of recognizing and responding to user misconceptions. The advisor confirms that a user's belief is a misconception by finding an explanation for why he does not hold the user's belief. The advisor infers its source by finding an explanation for the mistaken belief. The process of finding an explanation was presented as one of hypothesizing and trying to verify a small set of potential explanations associated with each type of user belief. In essence,

the model uses information about likely sources of different classes of user misconceptions to recognize user mistakes and infer their underlying causes.

This approach is attractive for several reasons. First, because it has information about classes of abstract misconceptions, it can handle misconceptions of which it has no prior knowledge, as long as they fall into one of these classes. A small set of potential explanations can account for a large number of specific user mistakes. Second, the model makes use of knowledge of the types of misconceptions users are likely to make to circumvent the need for general deductive reasoning. The model can easily be augmented to first check whether it has knowledge of specific misconceptions, as do tutoring systems, and to use general reasoning when it is confronted with a misconception that cannot be explained by any of its potential explanations, as does SPIRIT. Finally, our approach leads to responses similar (and sometimes more informative) than those of the UNIX advisors we have observed.

References

[1] Abelson, R. (1973). The Structure of Belief Systems. In R.C. Schank and K.M. Colby (Eds.), *Computer Models of Thought and Language*. Freeman, San Francisco, CA.

[2] Abelson, R. (1979). Differences between Beliefs and Knowledge Systems. *Cognitive Science*, 3. 355-366.

[3] Alvarado, S. (1987) *Understanding Editorial Text: A computer model of reasoning comprehension.* PhD Thesis. University of California, Los Angeles, CA.

[4] Alvarado, S., Dyer, M., and Flowers, M. (1986). Editorial Comprehension in OpED through Argument Units. *Proceedings of the 1986 National Conference on Artificial Intelligence.* Philadelphia, PA. 250-256.

[5] Anderson, J.R., Boyle, C.F., and Yost, G. (1985). The Geometry Tutor. *Proceedings of the 1985 Joint Conference on Artificial Intelligence.* Los Angeles, CA. 1-7.

[6] Brown, J.S. and Burton, R.R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, 2. 155-192.

[7] Burton, R.R. (1982). Diagnosing bugs in a simple procedural skill. *Intelligent Tutoring Systems*, Academic Press, London, England. 157-183.

[8] Carberry, S. (1987). Plan Recognition and User Modelling. *Computational Linguistics.* Special Issue on User Modelling.

[9] Chin, D. (1986). User modeling in UC, the UNIX consultant. *Proceedings of the CHI-86 Conference.* Boston, MA.

[10] Cullingford, S. (1978). *Script Application: Computer Understanding of Newspaper Stories*, PhD Thesis. Yale University, New Haven, CO.

[11] Dolan, C. and Dyer, M. (1985). Learning Planning Heuristics through Observation. *Proceedings of the 1985 Joint Conference on Artificial Intelligence.* Los Angeles, CA. 600-602.

[12] Dolan, C. and Dyer, M. (1986). Encoding Planning Knowledge for Recognition, Construction, and Learning. *Proceedings of the 8th Annual Cognitive Science Society.* 488-499.

[13] Dyer, M. (1983). *In-Depth Understanding: A Computer Model of Narrative Comprehension.* MIT Press. Cambridge, MA.

[14] Dyer, M., Flowers, M., and Reeves, J.F. (1987). Recognizing Situational Ironies: A computer Model of Irony Recognition and Narrative Understanding. To appear in *Advances in Computing and the Humanities.*

[15] Fikes, R.E. and Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2. 189-208.

[16] Flowers, M., McGuire, R., and Birnbaum, L. (1982). Adversary Arguments and the Logic of Personal Attacks. In *Strategies for Natural Language Processing.* Lawrence Erlbaum, Hillsdale, NJ. 275-294.

[17] Goodman, B. (1986). Miscommunication and Plan Recognition. Unpublished manuscript from UM86. International Workshop on User Modeling, Maria Laach, West Germany.

[18] Hammond, K. (1984). *Indexing and Causality: The Organization of Plans and Strategies in Memory.* Technical Report 351. Yale University, New Haven, CO.

[19] Kautz, H, and Allen, J. Generalized Plan Recognition. *Proceedings of the 1986 National Conference on Artificial Intelligence.* Los Angeles, CA. 423-427.

[20] Kemke, C. (1985). SC: Ein intelligentes hilfesystem fuer SINIX. (An intelligent help system for SINIX.) *LDV-Forum*, 2. 43-60.

[21] Joshi, A., Webber, B., Weishedel, R. (1984). Living up to expectations: computing expert responses. *Proceedings of the 1984 National Conference on Artificial Intelligence.* Dallas, TX. 169-175.

[22] Kolodner, J.L. (1984). *Retrieval and Organizational Strategies in Conceptual Memory.* Lawrence Erlbaum, Hillsdale, NJ.

[23] Kolodner, J.L. and Cullingford, R.E. (1986). Towards a Memory Architecture that Supports Reminding. *Proceedings of the 8th Annual Cognitive Science Society.* 467-477.

[24] McCoy, K. (1987). Reasoning on a dynamically highlighted user model to respond to misconceptions. *Computational Linguistics.* Special Issue on User Modelling.

[25] McCoy, K. (1985). Responding to Object-Oriented Misconceptions. Phd Thesis. University of Pennsylvania, Philadelphia, PA.

[26] Nesser, E. (1986). *SCUM: User modeling in the SINIX consultant.* Unpublished manuscript. University of Saarbrucken, Saarbrucken, West Germany.

[27] Paris, C. (1987). Tailoring object descriptions to the user's level of expertise. *Computational Linguistics.* Special Issue on User Modeling.

[28] Pollack, M. (1986). A model of plan inference that distinguishes between the beliefs of actors and observers. *Proceedings of 24th meeting of the Association of Computational Linguistics.* New York, NY.

[29] Pollack, M. (1986). *Inferring domain plans in question-answering.* PhD Thesis. University of Pennsylvania, Philadelphia, PA.

[30] Quilici, A. (1987). AQUA: A system that detects and responds to user misconception. In *User Modeling and Dialog Systems.* Springer Verlag, New York, NY.

[31] Quilici, A., Dyer, M., and Flowers, M. (1986). AQUA: An intelligent UNIX advisor. *Proceedings of the 1986 European Conference on Artificial Intelligence.* Brighton, England.

[32] Quilici, A. (1985). Human problem understanding and advice giving: A computer model. Technical Report #85-00069. Computer Science Department. University of California, Los Angeles, CA.

[33] Rees, J.A., Adams, N.L., and Meehan, J.R. (1984). *The T manual.* Yale University, New Haven, CT.

[34] Sacerdoti, E. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2). 115-135.

[35] Schank, R.C. (1982). *Dynamic Memory.* Cambridge University Press, Cambridge, MA.

[36] Schank, R.C., and Abelson, R. (1977). *Scripts, Plans, Goals, and Understanding.* Lawrence Erlbaum, Hillsdale, NJ.

[37] Schank, R.C. (1986). *Explanation Patterns: Understanding Mechanically and Creatively.* Lawrence Erlbaum, Hillsdale, NJ.

[38] Stevens, A., Collins, A., and Goldin, S.E. (1982). Misconceptions in students' understanding. *Intelligent Tutoring Systems*, Academic Press, London, England. 13-24.

[39] Turner, S.R. and Reeves, J.F. (1987). *The Rhapsody User's manual.* Technical Note UCLA-AI-86-10. Artificial Intelligence Laboratory, University of California, Los Angeles, CA.

[40] Wilensky, R., Mayfield, J., Albert, A., Chin, D., Cox, C., Luria, M., Martin, J., and Wu, D. (1986) *UC: A Progress Report.* Technical Report UCB/CSD 87/303. Computer Science Division (EECS), University of California, Berkeley, CA.

[41] Wilensky, R., Arens Y., and Chin, D. (1984). Talking to UNIX in English: An Overview of UC, *Communications of the ACM.* 574-593.

[42] Wilensky, R. (1983). *Planning and Understanding.* Addison Wesley, Reading, MA.

[43] Wilensky, R. (1978). *Understanding Goal-Based Stories.* Phd Thesis. Technical Report 140. Yale University, New Haven, CT.