The Correction Machine: Formulating Explanations For User Misconceptions

Alex Quilici
Artificial Intelligence Laboratory
Computer Science Department
University of California
Los Angeles, CA, 90024

Abstract

Novice computer users have many incorrect beliefs about the commands on their system. This paper considers the problem of providing explanatory responses that correct these mistaken user beliefs. Current approaches correct mistaken beliefs by trying to infer the reasons why the user holds them. In contrast, our advisor corrects these beliefs simply by explaining why he doesn't share them. This allows the advisor to provide reasonable advice even when no robust user model is available. Our advisor constructs this explanation from scratch, using a set of domain-independent strategies for justifying plan-oriented beliefs. This differs from existing systems, such as explanation-based story understanders, that provide explanations by modifying existing explanations and fail to address the underlying problem of forming the initial explanation. This approach gives our advisor the ability to explain novel misconceptions.

1 The Problem

Intelligent user interfaces and advisory systems must be able to provide explanations for mistaken user beliefs. Consider this dialog between a novice UNIX user and a more experienced UNIX advisor.

USER: I accidentally removed a file. How can I restore it now?
ADVISOR: YOU can't. It takes at least two days to restore a file.
USER: But I need the file today.
ADVISOR: YOU need to take care not to remove files you need. Why don't you use "rm -i" so that you're asked before any files are removed?
USER: It takes too long to answer the questions.
ADVISOR: It takes a long time to restore a file. But as a faster alternative to "rm -i", you could move unwanted files into "/tmp".
USER: But that's going to fill up "/tmp".
ADVISOR: "/tmp" is automatically cleaned up every few hours so filling it up isn't a problem.

This user has a set of mistaken beliefs that need to be corrected. These are beliefs that:

1. There's a plan that can restore files immediately.
2. The user's plan for removing files is appropriate.
3. Using "rm -i" isn't an appropriate plan for removing files because it takes too long to use.
4. Moving files to "/tmp" isn't an appropriate plan for removing files because it fills up "/tmp".

Each advisor response explains why one of these user beliefs is mistaken. The advisor explains that:

1. There is no way to recover files today because the plan for recovering files takes at least two days.
2. Using "rm -i" is an appropriate way to remove a file because it prevents accidental file removal.
3. The user's plan for removing files is inappropriate because it takes just as long as "rm -i" when the time to recover a file is taken into account.
4. Moving files to "/tmp" is an appropriate plan because there's a mechanism that automatically ensures that "/tmp" won't fill up.

How can an automated advisor produce these explanations? Out of the many beliefs the advisor possesses about UNIX, how does this advisor somehow manage to select a small set of beliefs that constitutes a reasonable explanatory response? This paper addresses these questions under the assumption that the advisor is initially presented with a representation for the user's mistaken beliefs. This assumption implies that the user's plans and goals have been recognized, a task addressed by many current systems [Carberry, 1988, Kautz and Allen, 1986, Wilensky, 1983].

2 Producing Explanatory Responses

The advisor can take one of two general approaches to responding to a user misconception.

The first is to try to understand why the user holds that incorrect belief, and to then address those underlying misconceptions. Why, for example, does this user believe that there's a way to quickly recover a removed file? One explanation is that the user usually works with PCs and is used to having immediately accessible backups on floppy disks. Another is that the user's last system had immediately accessible online backups. The advisor's task is to choose the most appropriate from all of these different explanations. The problem with this approach is that the advisor needs information about the user other than the user's explicitly stated beliefs. He has to know that the user usually works with PCs, or that most other users with this misconception previously worked with systems that had online backups. Often, however, this extra information isn't available, such as when it's the first time the advisor has encountered a particular misconception and a particular user.

The second approach is useful when the advisor has little information about the user. Rather than trying to understand the underlying causes of the user's mistake, the advisor tries to explain why he disagrees with the user's belief. In essence, the advisor simply tells the user, "here's why I think you're wrong." And that's exactly what the advisor in our initial example repeatedly does. The first belief the advisor corrects is the user's belief that there's a plan for rapidly recovering a file. To produce an explanation, the advisor tries to justify his belief that there is no plan for doing this task. The justification he finds is his belief that the plan for recovering files has some effect that causes it take at least two days. Similarly, the second belief the advisor corrects is the user's belief that he's using an appropriate plan for removing a file. The advisor tries to justify his belief that the user's plan is inappropriate. And the justification he finds is his belief that there's another plan, using "rm -i", that doesn't allow accidental file removal.

The remainder of this paper assumes that the advisor is taking this latter approach—that he's trying to correct mistaken user beliefs by explaining why he disagrees with them. We provide a model of how the advisor can formulate appropriate justifications for his contradictory beliefs. We concentrate on user beliefs involving plan applicability conditions, enablements, and effects.

3 Explanation-Based Belief Justification

Our approach for corning up with belief justifications is closely related to recent work in explanation-based story understanding [Schank, 1986]. These systems try to explain why a particular event occurred. Why, for example, might a racehorse have died within a week of winning a major race [Leake, 1988, Kass, 1986]? Their solution is to take an explanation for a similar, already-explained event, modify it to reflect the current event, and confirm that the explanation holds. In racehorse example, the system begins with its explanation for why the runner Jim Fixx died young: his jogging aggravated a congenital heart defect and led to a heart attack. The system modifies this explanation to use racing rather than jogging, and tries to confirm its new explanation that the horse had a congenital heart defect and that it died of a heart attack.

How does the system confirm the explanation? Essentially, it does one of two things. One is to examine whether pieces of the explanation match or conflict with known facts. Does it know that the horse had a heart attack? Or that the horse didn't have a congenital heart defect? The other is to test for predisposing circumstances that make this explanation more or less likely. One predisposing circumstance for a heart attack is that the victim is high-strung and easily excitable. Was this horse known to be high strung? The system, however, stops short of doing more general inferencing. It doesn't, for example, try to find evidence for whether the horse was easily excitable.

3.1 Our Approach To Justifying Advisor Beliefs

Our advisor's task is similar. These systems are trying to find and confirm an explanation for why an event occurred; our advisor is trying to find and confirm an explanation for why a belief is held. Our approach, however, differs in several significant ways. First, our advisor does not assume an explanation for a similar belief. Instead, the advisor starts from scratch, using a set of abstract belief-justification strategies to construct possible explanations. The feature of this approach is that it works without requiring an existing explanation.

And second, our advisor expends more effort confirming this explanation. The advisor doesn't accept an explanation for why a belief is held merely because nothing contradicts it, or because there are certain predisposing circumstances Instead, each piece of an explanation not directly confirmable by facts from memory must itself be explained. The advisor's explanation for why the user's file removal plan is inappropriate is that it doesn't prevent accidental file removal, even though "rm -i", another plan for file removal, does. To confirm this explanation the advisor is forced to explain how "rm -i" prevents accidental file removal, which happens to be because it asks the user before removing the file. Carefully confirming explanations is especially important when giving advice. Unconfirmed explanations are more likely to be wrong and mislead the user. And the additional information needed to confirm the explanation, such as "rm -i" asking a question, often forms useful advice.

3.2 Justification Patterns

Since we don't start with an explanation, how does the advisor explain why he holds a particular belief? We rely on a set of domain-independent strategies, *justification patterns* (JP)s, that represent potential explanations for why a particular belief is held. Each type of plan-oriented belief (that no plan is applicable to a goal, that a particular plan is applicable to a goal, that a particular state is caused by a particular plan, and so on) is associated with a set of JPs. A JP is an abstract pattern of planning relationships. Given a specific belief, the advisor classifies the belief, and then tries to instantiate and confirm one of the JPs associated with that class of belief. This instantiated and confirmed JP constitutes the contents of the advisor's response to the user.

This approach raises several questions. What justification patterns are needed for plan-oriented beliefs? How do we represent them? And how are they confirmed?

4 Representing Advisor Knowledge

The advisor must be able to represent two types of knowledge. The first involves the specific plan-oriented beliefs of the user and the advisor, such as those about file removal and recovery. What are some different ways to remove a file? What are the enablements of removing a file? What are the effects of using "rm -i"? And so on. The second is general knowledge about how to justify plan-oriented beliefs: the advisor's justification patterns.

4.1 Representing Plan-Oriented Beliefs

We represent beliefs in the same way as do most other systems that deal with the possibly contradictory beliefs of multiple dialog participants [Flowers et a/., 1982, Pollack, 1986a]. There's a relationship, beliej {A, R), that indicates that an actor A (either the user or the advisor) believes a planning relationship R holds.

Our representation for planning relationships combines elements used to represent plan effects and enablements found in other systems dealing with plan-oriented misconceptions [Pollack, 1986b], and elements needed to represent the goals and intentions of the user [Dyer, 1983]. We use the following set of primitive planning relationships. Here, A denotes an actor, S denotes a state (a description of properties of objects), P denotes a plan (a sequence of operators that, when executed, effects a state change), G denotes an actor's goal (a state an actor wants to achieve), and E denotes an event (an actor's execution of a particular plan).

A execute P	A carries out steps in P
E causes S	E has 5 as one of its effects
E leadsto E'	E has E' as one of its effects
S enables R	S is needed for relation R to hold
E applies G	E should be executed to achieve G
R precludes R'	R' can't hold if R holds
A has goal S	State S is a goal of actor A
R isa R'	R falls in the class R'
never R	No instantiation of R holds

These relationships provide a way to represent beliefs about a plan having a particular state as an enablement or effect, a plan being appropriate for a particular situation, and so on. Each of these relationships also has a corresponding negated relationship: that a plan doesn't cause a particular state, that a particular state is not an enablement to a plan, and so on. These sorts of beliefs are frequently found in advice-seeking dialogs. Our user, for example, believes that using "rm -i" is not a correct or normal plan for removing a file. And the advisor believes that there isn't some plan for recovering a file immediately.

A more complete description of the semantics of these relationships and a set of examples of the dialogs they represent can be found in [Quilici, 1989]. Here, however, we illustrate their semantics by using them to represent the user and advisor beliefs that are explicitly stated in our example dialog.

The user begins by providing a set of his beliefs: the user did execute a plan that causes a file to be removed, the user can execute some plan that applies to the goal of recovering the file today, and the user hasgoal to recover the file today. The advisor's response points out several corresponding advisor beliefs: the plan that applies to the goal of recovering a file has an effect that causes it to take two days before the file reappears, an effect that precludes recovering the file today.

The user's next response again provides a single user belief: the user hasgoal of accessing the file today. The advisor responds with another collection of advisor beliefs: the user's plan causes the user to have the unachievable goal of recovering the file today, "rm -i" applies to the goal of removing a file, and "rm -i" leadsto a question being asked.

The user follows up with his belief that answering questions causes him to use an excessive amount of time. The advisor responds with his belief that restoring files causes the same effect, and that moving files to "/tmp" applies to the goal of removing a file.

In the final exchange the user points out his belief that moving files to "/tmp" causes "/tmp" to fill up, and the advisor replies with his belief that the system frequently executes an action that causes "/tmp" to be cleaned up.

4.2 Representing Justification Patterns

The other knowledge the advisor needs is a set of justification patterns. There is a set of JPs for each of the planning relationships discussed in the previous section. Each JP is represented in terms of these same planning relationships. For example, one JP provides a potential justification for a belief that an actor A's executing a particular plan P doesn't apply to a particular goal G:

PLAN CAUSES UNACHIEVABLE GOAL

(1) A executing P causes A to have goal Gl
(2) There's no plan that applies to this goal G1
(3) A executing some other plan P' causes G
(4) A executing P' doesn't cause A to have G1

This JP suggests that an actor shouldn't use a particular plan to achieve a goal if it gives rise to another, unachievable, goal.

The advisor can use this JP to justify his belief that the user's plan for removing files is inappropriate. To do so, the advisor instantiates it with A as the user, P as the user's plan for removing files, and G as the user's goal of removing a file.

(1) The user executing his plan for removing a file causes him to have goal GI.
(2) There's no plan that applies to this goal GI.
(3) The user executing some other plan P'causes the file to he removed.
(4) The user executing P'doesn't cause him to have GI.

The advisor then tries to confirm this partially instantiated JP.

4.3 Confirming Justification Patterns

Flow does the advisor confirm one of these JPs? The advisor repeatedly selects one of its belief to confirm, tries to confirm it, and instantiates the JP with any new information gleaned from the confirmation process. To confirm a particular belief, the advisor searches memory for a matching belief (or specific instances of it) and, if that fails, then the advisor tries to justify it using additional JPs. This process stops when the JP is successfully instantiated and confirmed, or when memory search fails to yield new confirming beliefs.

For the above JP, the advisor first tries to confirm that the user has some goal *G1* Memory search yields the confirming belief that the user has a goal to recover a file today, a goal stated in the user's first utterance, and the advisor instantiates the JP with this information.

(1) The user executing his plan for removing a file causes him to have a goal of recovering the file today.
(2) There's no plan that applies to this goal of recovering the file today.
(3) The user executing some other plan P' causes the file to be removed.
(4) The user executing *P'*doesn't cause him to have a goal of recovering a file today.

The advisor then repeats the process for each of these subsequent beliefs. The result of the confirmation process is a fully instantiated and confirmed JP.

Here we've assumed that confirming beliefs are readily found. But what if the advisor didn't already have a belief in memory that ''rm -i" doesn't cause the user to have a goal of recovering a file? In that case, he would have had to try to justify holding this belief. Fortunately, he can use this same method. This belief falls into the class that an actor A'S executing a plan *P* doesn't cause a particular effect 5. One JP for this class suggests that the plan leads to an action that has another effect that precludes this effect.

PLAN INDIRECTLY PRECLUDES EFFECT

(1) *A* executing the plan Pleads to an event *E*
(2) *E* causes *S'*
(3) State *S'* precludes *S*

The advisor instantiates this JP with *P* as "rm -i" and *S* as the user having a goal of removing the file. The advisor then tries to confirm these beliefs, leading to the final justification:

(1) The user executing the plan of using "rm -i" leads to the system asking the user before removing the file.
(2) Asking questions causes the file to still exist.
(3) The file still existing precludes the user having the goal of recovering the file.

The advisor then includes these beliefs in his response.

4.4 Some Other Justification Patterns

Justification patterns capture knowledge about belief justifications that is independent of any particular plan. That means that each JP can be used to form justifications for many different beliefs. In our introductory dialog, for example, a single JP is used by both the user and the advisor to justify their differing beliefs about which plan the user should execute to achieve the goal of removing a file. This JP suggests that a plan doesn't apply to a goal if it leads to an action that thwarts a different goal, while another plan for that goal doesn't lead to that action.

PLAN INDIRECTLY THWARTS GOAL

(1) *A* executing Pleads to executing *Po*
(2) Executing *Po* causes state *S*
(3) *A* hasgoal S'
(4) *S* precludes *S'*
(5) *A* execute P' causes *G*
(6) *A* execute *P'* doesn't lead to *Po*

The user uses this JP to justify his belief that using "rm -i" doesn't apply to the goal of removing a file. The user instantiates it as:

(1) The user executing "rm -i" leads to his being asked a question before removing the file.
(2) His being asked questions causes use extra time.
(3) The user has a goal to preserve available time.
(4) Using extra time precludes preserving available time.
(5) The user's file removal plan causes the file's removal.
(6) The user executing his removal plan doesn't lead to his being asked questions.

In other words, "rm -i" violates a user goal by leading to a time-consuming question that the user's file removal plan doesn't lead to. The advisor uses this same JP to justify his belief that the user's file removal plan doesn't apply to goal of removing a file. His response instantiates this JP as:

(1) The user executing ''rrn -i" leads to a file recovery action.
(2) Executing the file recovery action causes a long wait.
(3) The user has a goal to preserve available time.
(4) A long wait precludes preserving available time.
(5) The user executing "rm -i" applies to removing fde.
(6) The user executing ''rm -i" doesn't lead to a fde recovery action.

That is, the user's file removal plan violates a user goal by leading to a time-consuming action to recover the file that "rm -i" doesn't lead to.

Each type of belief has a collection of justification patterns. Our example contains several other JPs for the belief of a plan not applying to a goal. One JP the advisor uses suggests that a plan doesn't apply to a goal if one of its effects violates an enablement of a plan for another goal.

PLAN DISABLES PLAN FOR OTHER GOAL

(1) *A* executing plan P^f causes *G*
(2) *A* executing *P'*causes *S*
(3) *A* hasgoal *G'*
(4) *S* enables P_o causes 6"
(5) *A* execute *P* causes S'
(6) S'precludes 5

The advisor instantiates it to further justify the belief that the user is using the wrong plan for removing files:

(1) The user executing the plan of moving files to ^{a}/tmp" causes the file to be removed.
(2) The user executing the plan of moving files to "/tmp" causes the file to exist in ^{u}/tmp".
(3) The user hasgoal of recover file today.
(4) The file existing in ^{tt}/^{trn}P" enables recovering the file by executing the plan of moving files from ''/tmp".
(5) The user execute removal plan causes file not exist.
(6) The file not existing precludes it existing in ''/tmp".

The justification is that the plan of moving files to "/tmp" achieves an enablement of the recovery plan of restoring the files from "/tmp" (namely, that the file exists in "/tmp"); the user's removal plan keeps this enablement from being achieved.

One final JP for this belief type appears in the user's last response. The user uses it to support his belief that moving files to "/tmp" doesn't apply to the goal of removing a file. This JP says that a plan isn't applicable to a goal if it has an effect that thwarts another goal and another plan for the goal doesn't have this effect.

PLAN THWARTS GOAL

(1) *A* execute *P'* causes *G*
(2) *A* execute *P* causes *S*
(3) *S* precludes *S'*
(4) *A* hasgoal *S'*
(5) *A* execute *P'* doesn't cause *S*

The user instantiates it as:

(1) The user executing his file removal plan causes the file to be removed.
(2) The user executing move to "/tmp" causes "/tmp" to fill up.
(3) "/tmp" filling up precludes having space in "/tmp".
(4) The user has a goal of having space in "/trnp".
(5) The user executing his file removal plan doesn't cause "/tmp" filling up.

That is, moving files to "/tmp" causes it to fill up, violating the user goal of having space in "/tmp", an effect the user's plan doesn't have.

5 Implementation Status

The model discussed in this paper has been implemented in a Prolog program (the Correction Machine). The program's domain of expertise is the basic UNIX commands needed to remove, recover, and rename files. It also possesses approximately 25 different justification patterns spanning the planning relationships discussed in this paper. The input to the program is a representation for a user belief involving plan applicability conditions, enablements, or effects. The program's output is the set of advisor beliefs to present to the user, along with a description of the particular justification patterns needed to produce the explanation. It can formulate explanatory responses corresponding to several different dialogs, including our initial example.

Currently, we're trying to answer several questions about our model. First, how well do the justification patterns described here account for responses to misconceptions in domains other than those of novice computer users? To test their domain-independence, we're extending the program to give advice about simple day-to-day planning. Second, how sufficient is our set of justification patterns for providing UNIX advice? We're now studying many different user/advisor dialogs, searching for the presence of other useful justification patterns.

We're also working on more quickly finding appropriate justifications, as this process is potentially time-consuming, especially when confirming a single justification pattern may involve trying to confirm many other JPs. There are two ways to speed up the process; both of which lead to unanswered questions. One is to save instantiated justification patterns for later use. But how are these specific JPs organized and retrieved? The other is to pick JPs that are likely to be confirmed successfully. But how can the most appropriate JP be selected?

Finally, we're trying to extend the model toward being a more complete dialog participant, a task that raises several other important questions. First, can justification patterns be used to *understand* belief justifications? A dialog participant must not only construct his own belief justifications but also understand those of others. Ideally, the same knowledge should be used by both processes. But our model is solely concerned with constructing belief justifications, not comprehending them. And second, how is the particular belief to justify selected? A dialog participant quickly faces a large collection of beliefs from which he must choose a particular belief to justify. Our model, however, simply assumes that this belief has already been chosen.

6 Comparison With Related Work

There are two types of related systems: those that provide advice to novice UNIX users, and those that correct misconceptions. Existing UNIX advisors such as UC [Wilensky *et a/.*, 1986] and SC [Hecking *et a/.*, 1988] don't attempt to explain mistaken beliefs. Instead, they assume the user's problem is incomplete knowledge, and focus on filling in the gaps indicated by questions such as "Ilow do I remove a file?" or "What does 'rm -i' do?".

One system that does try to explain user misconceptions is our own earlier effort, AQUA [Quilici, 1989, Quilici *et a/.*, 1988, Quilici *et a/.*, 1986]. This system corrects plan-oriented misconceptions using strategies similar to our justification patterns. A problem with AQUA is that its confirmation process is limited to memory search; it makes no attempt to justify holding the beliefs it's trying to confirm. This severe limitation arises because AQUA has strategies only for only why it doesn't believe something, and not for why it does. This means that AQUA has no way to support a belief that a plan does apply to a goal, or that a plan does have a particular effect, beliefs that may have to be justified for a strategy to be confirmed.

ROMPER [McCoy, 1988] applies an approach similar to AQUA's to correct a different class of misconceptions: mistaken user beliefs involving misclassifications or misattributions involving objects. ROMPER also has a set of strategies for justifying its beliefs to the user. But because ROMPER deals only with two different kinds of mistaken beliefs and has only a few strategies, it suffers from the same drawbacks as AQUA. It's forced to abandon a strategy if it doesn't find confirming evidence when it searches memory.

The final system, SPIRIT [Pollack, 1986a, Pollack, 1986b], detects the use of inappropriate plans of users of a computer mail program, along with the mistaken user beliefs underlying those plans. SPIRIT's task is similar to ours, but it takes a completely different approach. It uses rules to try to infer what the user's beliefs actually are, rather than justifying the advisor's own beliefs.

One such rule 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. Because it takes this approach, SPIRIT does not specify what information should be included in a cooperative response, something that falls naturally out of our model.

Finally, there's a large collection of tutoring systems [Sleeman and Brown, 1982] that attempt to correct and explain user misconceptions. Generally, these systems locate mistaken beliefs in a database of domain-specific error-explanation pairs and provide the associated explanation. This approach has several drawbacks. Because the explanations are domain-specific, having the tutor provide explanations for mistakes in a new domain involves finding a new set of error-explanation pairs. And because these systems simply retrieve explanations, they can handle only those misconceptions they know about in advance.

7 Conclusions

This paper makes two contributions. First, we suggest a novel approach to formulating responses that correct mistaken user beliefs. Rather than trying to infer what led the user astray, our advisor simply says why the user is wrong. This allows the advisor to provide helpful responses, even when no robust user model is available. And second, we propose a mechanism for constructing explanations from scratch. Our advisor explains why he holds a particular belief by using a set of domain-independent strategies for justifying plan-oriented beliefs. These strategies constrain the memory search for appropriate supporting beliefs, limiting it to those beliefs that are likely to be useful. And they provide a way to justify beliefs that memory search fails to locate. This ability to create novel explanations out of domain-independent strategies can be used to form those initial explanations needed by systems that work by modifying existing explanations.

Acknowledgments

Special thanks go to Edward Hoenkamp for his many helpful hints on how to improve this paper, as well as to Michael Dyer and the anonymous referees for their comments and suggestions.

References

[Carberry, 1988] S. Carberry. Modeling the user's plans and goals. Computational Linguistics, 14(3):23-37, September 1988.

[Dyer, 1983] M.G. Dyer. In-depth understanding: A computer model of narrative comprehension. MIT Press, Cambridge, MA, 1983.

[Flowers et al., 1982] M. Flowers, R. McGuire, and L. Birnbaum. Adversary arguments and the logic of personal attacks. In Strategies for Natural Language Processing, pages 275-294. Lawrence Erlbaum, Hillsdale, NJ, 1982.

[Hecking et ai, 1988] M. Recking, C. Kemke, E. Nessen, D. Dengler, M. Gutmann, and G. Hector. The SINIX Consultant — A progress report. Technical Memo 28, University of Saarbrucken, Saarbrucken, West Germany, 1988.

[Leake, 1988] D.B. Leake. Evaluating explanations. In Proceedings of the Eighth National Conference on Artificial Intelligence, pages 251-255. Saint Paul, MN, August 1988.

[Kass, 1986] A. Kass. Modifying explanations to understand stories. In Proceedings of the Eighth Annual Conference of the Cognitive Science Society, pages 691-696. Amherst, MA, 1986.

[Kautz and Allen, 1986] H. Kautz and J. Allen. Generalized plan recognition. In Proceedings of the Sixth National Conference on Artificial Intelligence, pages 423-427. Philadelphia, PA, 1986.

[McCoy, 1988] K. McCoy. Reasoning on a highlighted user model to respond to misconceptions. Computational Linguistics, 14(3):52-63, September 1988.

[Pollack, 1986a] M. Pollack. A model of plan inference that distinguishes between the beliefs of actors and observers. In Proceedings of 24th meeting of the Association of Computational Linguistics, pages 207-214, New York, NY, 1986.

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

[Quilici, 1989] A. Quilici. AQUA: A system that detects and responds to user misconceptions. In User Modeling and Dialog Systems. Springer Verlag, New York, NY, 1989.

[Quilici et ai, 1988] A. Quilici, M.G. Dyer, and M. Flowers. Recognizing and responding to plan-oriented misconceptions. Computational Linguistics, 14(3):38-51, September 1988.

[Quilici et ai, 1986] A. Quilici, M.G. Dyer, and M. Flowers. AQUA: An intelligent UNIX advisor. In Proceedings of the 1986 European Conference on Artificial Intelligence, Brighton, England, July 1986.

[Schank, 1986] R. Schank. Explanation patterns: Understanding mechanically and creatively. Lawrence Erlbaum, Hillsdale, NJ, 1986.

[Sleeman and Brown, 1982] D. Sleeman and J.S. Brown, Editors. Intelligent Tutoring Systems. Academic Press, Orlando, FL, 1982.

[Wilensky et ai, 1986] R. Wilensky, D. Chin, M. Luria, J. Martin, J. Mayfield, and D. Wu. The Berkeley UNIX Consultant Project. Computational Linguistics, 14(4):35-84, December 1988.

[Wilensky, 1983] R. Wilensky. Planning and understanding. Addison Wesley, Reading, MA, 1983.