Artificial Intelligence Review **14:** 403–420, 2000.
Stephen J. Hegner et al. (eds.), Intelligent help systems for UNIX.

Using Justification Patterns to Advise Novice UNIX Users

ALEX QUILICI*
Department of Electrical Engineering, University of Hawaii at Manoa, 2540 Dole St, Holmes 483, Honolulu, HI, 96822, USA (E-mail: alex@wiliki.eng.hawaii.edu)

Abstract. Novice UNIX users have many incorrect beliefs about UNIX commands. An intelligent advisory system for UNIX should provide explanatory responses that correct these mistaken beliefs. To do so, the system must be able to understand how the user is justifying these beliefs, and it must be able to provide justifications for its own beliefs. These tasks not only require knowledge about specific UNIX-related plans but also abstract knowledge about how beliefs can be justified. This paper shows how this knowledge can be represented and sketches how it can be used to form justifications for advisor beliefs and to understand justifications given for user beliefs. Knowledge about belief justification is captured by justification patterns, domain-independent knowledge structures that are similar to the abstract knowledge structures used to understand the point behind a story. These justification patterns allow the advisor to understand and formulate novel belief justifications, giving the advisor the ability to recognize and respond to novel misconceptions.

Keywords: belief inference, plans, user modeling

1. The Problem

An intelligent UNIX help system should be able to provide explanations that correct mistaken plan-oriented user beliefs. Consider this dialog between a novice UNIX user and a more experienced UNIX advisor.

(1) USER: I accidentally removed a file using “rm”. How can I restore it now?
(2) ADVISOR: You can’t. It takes several days to restore a file.
(3) USER: But I need to access the file now.
(4) ADVISOR: You should use “rm -i” to remove files. That way you’re asked before any files are removed.
(5) USER: But I don’t want to be asked questions.
(6) ADVISOR: You could move unwanted files into “/tmp”.
(7) USER: But that’s going to fill up “/tmp”.
(8) ADVISOR: “/tmp” is automatically cleaned up every few hours, so filling it up isn’t a problem.

This user has this set of mistaken beliefs that need to be corrected:

- There is a plan that can restore files immediately.
- "rm" is an acceptable plan for removing files.
- Removing files to "/tmp" is not an acceptable plan for removing files.
- Removing files to "/tmp" fills up "/tmp".

The advisor's responses explain why these user beliefs are mistaken. The advisor explains that:

- There is no way to recover files today because the action for recovering files takes at least two days.
- "rm -i" should be used instead of "rm" because it asks before removing the file.
- Removing files to "/tmp" is an acceptable plan for removing files because because it does not fill up "/tmp".
- Removing files to "/tmp" does not fill up "/tmp" because some automatic action removes the files in that directory.

To provide cooperative explanations like these, the advisor must be able to construct a model of the user's beliefs and their justifications, and the advisor must be able to formulate justifications for why various user beliefs are incorrect. This paper is concerned with the knowledge the advisor needs to construct this user model and how this knowledge is represented and used. It assumes that the advisor is initially presented with a representation for the beliefs underlying a given user's utterance. This assumption implies that the user's plans and goals have been recognized, a task addressed by many current systems (Carberry 1989; Kautz and Allen 1986; Wilensky 1983).

2. What the Advisor Needs to Know

A UNIX advisory system needs to represent two distinct classes of knowledge.

The first involves specific plan-oriented user and advisor beliefs. In our dialog, for example, the user and advisor provide a sizeable set of beliefs about the various plans available for removing and recovering files. These beliefs are concerned with which plan is most appropriate for a given goal, whether or not there is a plan that achieves a particular goal, whether a particular state is an enablement or effect of a plan, what goals the user has, and so on.

The other involves knowledge about how to justify plan-oriented beliefs. The advisor needs this knowledge to construct the belief justifications that are presented to the user, such as the advisor's reason for why there is no plan that achieves the goal of recovering a file today. Furthermore, the advisor needs this knowledge to understand the user's belief justifications, such as the user's reasons for why "rm -i" and moving files to "/tmp" are not appropriate plans

Table 1. The planning relationships relevant to our example dialog.

RELATIONSHIP	SEMANTICS
A hasgoal *E*	*E* is a goal of actor *A*
E applies E'	*E* is an acceptable way to achieve *E*
E causes E'	*E* has E' as one of its effects
E interferes E'	E' cannot occur if *E* does
E enables E'	*E* is necessary for E' to occur
E preferred E'	*E* is more desirable than E'

for the goal of removing a file. It is clearly possible to capture some of this knowledge in specific belief justifications. The advisor could, for example, possess concrete knowledge that "rm -i" is an appropriate plan for removing a file because it asks the user before removing the file. Furthermore, the advisor might possess the knowledge that "rm -i" might be considered inappropriate because users don't want to be asked a question. The advisor, however, cannot be expected to possess all possible specific belief justifications in advance. To form these justifications the advisor clearly needs some general knowledge about what constitutes an appropriate justification.

2.1. *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 al. 1982; Pollack 1986a, 1986b). There is a relationship, *belief*(*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 1986a, 1986b) and elements needed to represent the goals and intentions of the user (Dyer 1983). We are concerned with the planning relationships shown in Table 1. There, *A* denotes an actor and *E* denotes an event (an actor's execution of a particular plan or an action that takes place as a result of a plan's execution).

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, *not-R*, which is used to represent beliefs such as a plan not having a particular enablement or effect. These sorts of beliefs

are frequently found in advice-seeking dialogs. The advisor, for example, believes that the file recovery plan does not causes the file to be restored immediately, and he user believes that "rm" does not lead to his being asked questions.

This small set of planning relationships can be used to represent the explicitly provided beliefs found in our example dialog.[1]

Each dialog utterance corresponds to a small set of beliefs.

- In (1), the user provides an initial set of beliefs. One is that executing "rm" *causes* a file to be removed. *causes* represents a plan effect: a state change or action that results from a plan's execution. Another user belief is that there is some plan that *causes* the file to be recovered immediately. The last user belief is that the user *hasgoal* of recovering the file immediately. *hasgoal* represents an actor desiring that a particular state hold or a particular event take place.
- In (2), the advisor's response provides a pair of beliefs. One is that there is no plan that *causes* the user's file to be recovered immediately. The other is that there is some plan that *causes* the file to be recovered eventually.
- In (3), the user provides one new belief: that the user *hasgoal* of accessing the file today.
- In (4), the advisor again provides a pair of beliefs. One is that using "rm -i" *applies* to the goal of removing a file. *applies* represents the notion that a plan is an acceptable way to achieve a particular goal. The other belief is that "rm -i" *causes* a question that precedes the file's being removed.
- In (5), the user follows up with a belief that the user *hasgoal* of not being asked questions.
- In (6), the advisor provides a single belief that moving the file to "/tmp" *applies* to the goal of removing a file.
- In (7), the user provides a single belief that moving files to "/tmp" *causes* "/tmp" to fill up.
- In (8), the advisor provides one final pair of beliefs. One is that moving files to "/tmp" *not-causes* "/tmp" to fill up. The other is that there is an action that *interferes* with "/tmp" filling up. The *interferes* relationship represents an action or effect preventing another action or effect from occurring.

The user and advisor also appear to hold other, related beliefs that are not explicitly provided as part of the dialog. In particular, several of the user and advisor responses implicitly suggest other beliefs the user holds.

Table 2. The JPs that appear in our example dialog.

RELATIONSHIP	JUSTIFICATION PATTERN
not-exists *some* E', where E' causes E	JP:CLASS-ACTION-THWARTS-EFFECT
A hasgoal E	JP:ENABLEMENT-FOR-GOAL
E applies E'	JP:PREFERRED-ALTERNATIVE
E not-applies E'	JP:VIOLATES-OTHER-GOAL
E preferred E'	JP:ACHIEVES-ADDITIONAL-GOAL
E interferes E'	JP:HAS-THWARTING-EFFECT
E not-causes E'	JP:THWARTED-BY-OTHER-ACTION

- In (3), the user appears to believe that restoring a file *enables* accessing the file. The *enables* relationship represents one action being necessary for another to occur.
- In (4), the advisor appears to believe that "rm -i" is *preferred* to "rm". The *preferred* relationship represents one action being more desirable than another action.
- In (5), the user appears to be believe that "rm -i" *not-applies* to the goal of removing a file.
- In (7), the user appears to believe that removing a file to "/tmp" *not-applies* to the goal of removing a file and that he *hasgoal* of not filling up "/tmp".

2.2. *Representing abstract knowledge about belief justification*

We represent abstract knowledge about how to justify different kinds of beliefs using a set of *Justification Patterns* (JPs). A JP is an abstract configuration of planning relationships that captures one class of belief justification for one type of planning relationship. Different types of planning relationships, such as one plan being preferred to another, or a plan having a particular effect, are associated with different JPs. Instantiated JPs form the justifications for holding specific beliefs.

Our example dialog makes use of a set of JPs for a variety of different belief types. Each response, in fact, makes use of at least one JP. Table 2 summarizes the different JPs used and the belief classes they justify.

Here, we run through those responses and show how each of those JPs are represented and where each is used.

- In (1), the user is simply describing his situation and not trying to justify any beliefs, so no JPs appear.

– In (2), the advisor uses a pair of JPs. The first, JP:CLASS-ACTION-THWARTS-EFFECT, justifies the advisor's belief that there is no plan for immediately recovering a file. In general, this JP provides a way to justify a belief that there is no plan that causes a particular effect *E* (restoring the file today). The justification is that an action that causes a more general effect (restoring the file) somehow interferes with *E* (restoring the file immediately).

JP:CLASS-ACTION-THWARTS-EFFECT
not-exists E', where E' causes *E*
| justifies
E isa *X*
P causes *X*
P interferes *E*

The other, JP:HAS-THWARTING-EFFECT, is used to justify the advisor's belief that a file recovery plan interferes with recovering the file today (the final belief in the advisor's instantiation of the preceding JP). In general, this JP is a way to justify a belief that executing a plan *P* (the recovery plan) interferes with an effect *E* (recovering the file today). The justification is that this plan *P* has another effect E' (taking two days to recover the file) that keeps *E* from occurring.

JP:HAS-THWARTING-EFFECT
P interferes *E*
| justifies
P causes E'
E' interferes *E*

– In (3), the user uses a single JP, JP:ENABLEMENT-FOR-GOAL, to justify his belief that he wants to recover the file immediately. In general, this JP can be used to justify a belief that an actor *A* (the user) has a goal *G* (recovering a file immediately). The justification is that *G* is an enabling condition for some other goal G' (accessing the file today).

JP:ENABLEMENT-FOR-GOAL
A hasgoal *G*
| justifies
G enables G'
A hasgoal G'

– In (4), the advisor again uses a pair of JPs. The first, JP:PREFERRED-ALTERNATIVE, justifies the advisor belief that "rm -i" should be used to remove files. In general, this JP is used to justify a belief that executing a plan *P* ("rm -i") applies to a goal *G* (removing a file). The justification is that *P* is preferred to another plan *P′* ("rm") that also achieves *G*.

JP:PREFERRED-ALTERNATIVE
P applies *G*
| justifies
P causes *G*
P′ causes *G*
P preferred *P′*

The other, JP:ACHIEVES-ADDITIONAL-GOAL, is used to justify the advisor's belief that "rm -i" is preferred to "rm" (the final belief in the advisor's instantiation of the preceding JP). In general, the JP is used to justify a belief that an action *P* ("rm -i") is preferred over another action *P′* ("rm"). The justification is that *P* achieves another goal *G* (asking questions before removing the file) that *P′* does not.

JP:ACHIEVES-ADDITIONAL-GOAL
P preferred *P′*
| justifies
A hasgoal *G*
P causes *G*
P′ not-causes *G*

– In (5), the user uses another new JP, JP:VIOLATES-OTHER-GOAL, to justify a belief that "rm -i" is not applicable to the user's goal of removing a file. In general, this JP is used to justify a belief that a plan *P* ("rm -i") is not applicable to a goal *G* (removing a file). The user's justification is that *P′* ("rm -i") has an effect *E* (asking questions) that the user has a goal to avoid.

JP:VIOLATES-OTHER-GOAL
P not-applies *G*
| justifies
P causes *G*
P causes *E*
A hasgoal *not E*

– In (6), the advisor simply provides a new alternative plan for the goal, with no justification.

– In (7), the user uses the same JP, JP:VIOLATES-OTHER-GOAL, used in (5), but this time uses it to justify a belief that removing a file by moving it to "/tmp" is an unacceptable plan. In particular, removing a file to "/tmp" fills up "/tmp", an action the user has a goal to avoid.
– In (8), the advisor uses one final JP, JP:THWARTED-BY-OTHER-ACTION. This JP justifies his belief that moving files to "/tmp" does not cause "/tmp" to fill up. In general, this JP is used to justify the belief a plan *P* (moving files to "/tmp") does not have an effect *E* (filling up "/tmp"). The justification is that there is some other action (the clean-up program) that interferes with *E*.

JP:THWARTED-BY-OTHER-ACTION
P not-causes *E*
| justifies
E′ interferes *E*
exists *P*′ causes *E*′

3. Using Justification Patterns

Why are justification patterns important? JPs capture knowledge about belief justifications that is independent of specific plans and can be used in a variety of situations, allowing novel belief justifications to be formulated and understood without the need for a large collection of specific belief justifications.

To see how this knowledge is useful, consider the advisor's processing of the beliefs in (5). There, the advisor is presented with a single user belief: namely, that the user has a goal not to be asked questions. The advisor's first task is to understand why the user is providing these beliefs. That involves figuring out which belief, if any, these beliefs justify, and which other user beliefs form an unstated component of this justification. In particular, the advisor must infer that the user is justifying a belief that "rm -i" is not appropriate for removing files, and the advisor must infer the unstated user belief in this justification: that "rm -i" leads to the user being asked questions.

How can the advisor can make those inferences? One way is to find the JP that captures the user's stated belief and that somehow relates this to previous user and advisor beliefs. The advisor can then assume that the user is trying to justify the belief justified by that JP and that the user holds the other beliefs within that JP.

The advisor's other task is to find an appropriate justification for any advisor beliefs that differ from the user's. For example, in (4), the advisor

presents “rm -i” as being preferred to “rm” for removing files and justifies it by noting that it asks the user before actually removing the file. How can the advisor construct this justification? By using one of the JPs for a belief that one plan is preferred to another. The JP guides the advisor’s memory search for appropriate justifying beliefs. When the advisor finds a set of beliefs that can successfully instantiate the JP, those beliefs constitute the advisor’s response to the user.

3.1. *Comprehending belief justifications*

How exactly are belief justifications understood?

The process of comprehending the user’s belief justifications breaks into several parts: selecting and instantiating a likely-to-be-relevant JP, relating it to a belief already in the dialog, and confirming that it is reasonable for the user to have used this JP in forming his response.

First, the advisor finds a candidate JP by running through each of the JPs that contain a relationship corresponding to one of the user’s newly-stated beliefs. In (5), for example, the advisor is presented with the belief that the user has a goal of not being asked questions. This belief matches a belief in JP:ENABLEMENT-FOR-GOAL, resulting in this instantiation of that JP:

“rm -i” not-applies *G*
  | justifies
“rm -i” causes *G*
“rm -i” causes *ask question*
user hasgoal *not asked question* (stated user belief)

Second, the advisor has to relate this candidate JP to some belief that has already been stated or inferred from the dialog. The advisor must see if this JP justifies a belief that matches a known user belief, negates a known advisor belief, or appears in a JP that justifies one of those beliefs. At this point, as far as the advisor knows, the user holds no matching belief. However, the negation of the belief justified by this JP matches this previously stated advisor belief:

“rm -i” applies *remove-file*

As a result, the advisor can instantiate the JP with this new information.

“rm -i” not-applies *remove-file* (contradicts stated advisor belief)
  | justifies
“rm -i” causes *remove-file*
“rm -i” causes *ask question*
user hasgoal *not asked question* (stated user belief)

Finally, the advisor must confirm that it is reasonable for the user to have used the candidate JP. That is, the advisor must verify that the user holds any beliefs in this JP that were not explicitly provided in the current response. This verification of a belief is done by either:

- Finding that the user provided those beliefs earlier in the dialog, or
- Determining that the advisor holds those beliefs and the user has provided no explicit contradiction.

Here, the advisor is trying to find user beliefs that "rm -i" causes file removal, that it causes a question to be asked, and that the user has a goal not to be asked questions. The user provides the last explicitly, and the advisor believes the first two and the user has provided no beliefs that contradict them. As a result, the advisor assumes the user holds these beliefs and has confirmed that the user is using this JP, and the advisor is now aware of which belief the user is justifying.

3.2. *Constructing belief justifications*

How exactly are belief justifications constructed?

The process of constructing belief justifications has several parts: selecting a candidate JP, instantiating it with information from the belief being justified, and then verifying that the advisor holds the beliefs in this partially-instantiated JP.

First, the advisor finds a candidate JP by classifying the belief to be justified and selecting one of the JPs corresponding to that class of belief. In our example, the advisor must construct a justification for the belief that "rm -i" should be used to remove a file. The advisor classifies this belief as a plan applying to a goal and selects one of the JPs associated with that type of belief: JP:PREFERRED-ALTERNATIVE.

Second, the advisor instantiates the candidate JP with information from the belief he is trying to justify. Here, the advisor instantiates the JP with *P* as "rm -i" and *G* as removing a file.

rm -i applies *remove file* (advisor belief to justify)
  | justifies
rm -i causes *remove file*
P′ causes *remove file*
rm -i preferred *P′*

Finally, the advisor then tries to confirm that the beliefs in this candidate JP form a suitable justification. The advisor does this by repeatedly selecting a belief in the JP, trying to verify it, and then instantiating the JP with any new information gleaned from the verification process. To verify a particular belief, the advisor searches memory for a matching belief (or specific

instances of it) and, if that fails, tries to justify the belief using additional JPs. This process stops when the JP is successfully instantiated and confirmed or when memory search fails to yield new verifying beliefs.

For the above JP, the advisor first tries to verify that "rm -i" removes a file. Memory search yields the confirming belief that it does. The advisor then tries to find a match for the belief that there is some other plan that can remove a file. In this case, the advisor locates the belief that "rm" does so and and the advisor instantiates the JP with this information.

rm -i applies *remove file* (advisor belief to justify)
| justifies
rm -i causes *remove file* (stated advisor belief)
rm causes *remove file* (stated user belief)
rm -i preferred *rm*

That leaves one belief to verify, that "rm -i" is preferred to "rm" for removing files. Unfortunately, the advisor doesn't locate this belief in memory. As a result, the advisor must try to justify this belief using JPs. In this case, the advisor uses JP:ACHIEVES-ADDITIONAL-GOAL to try to justify this belief, which the advisor instantiates as:

rm -i preferred *rm* (advisor belief to justify)
| justifies
user hasgoal *G*
rm -i causes *G*
rm not-causes *G*

This leaves the advisor several beliefs to try to instantiate and verify. In this case, the advisor locates the belief that *rm -i* causes a question to be asked before file removal, resulting in this JP being instantiated as:

rm -i preferred *rm* (advisor belief to justify)
| justifies
user hasgoal *asked before remove*
rm -i causes *asked before remove* (stored advisor belief)
rm not-causes *asked before remove*

The advisor now locates the remaining beliefs in memory: that *rm* does not ask before removing the file and that the user has a goal of being asked before the file is removed.

The result of the confirmation process is a fully instantiated and verified JP.

4. Implementation Status

The model discussed in this chapter has been implemented in a Prolog program: *The Correction Machine*. The program's current domain of expertise is the basic UNIX commands needed to remove, recover, and rename files. It possesses the seven justification patterns discussed here, along with ten others that we have found useful in processing variants of our example dialog. The program itself consists of a COMPREHENDER and a CONSTRUCTOR. The COMPREHENDER'S input is a representation for a set of user beliefs. Its output is the beliefs justified by these beliefs and the instantiated justification patterns to which these beliefs belong. In addition, the output shows the relationships between these beliefs and other beliefs already specified in the dialog. The CONSTRUCTOR'S input is a particular advisor belief. Its output is a set of beliefs that justify holding this belief, along with a description of the particular justification patterns the advisor used to produce that explanation. The program can construct and comprehend all the belief justifications in the file removal debate.

Currently, we are 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 are 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 are now studying many different user/advisor dialogs, searching for the presence of other useful justification patterns.

We are also working on improving our model's performance, particularly on the task of selecting potentially useful JPs, as this process is potentially time-consuming. During comprehension, for example, an input belief may correspond to a variety of different JPs, many of which are eventually ruled out because they can't be related to other beliefs in the dialog or happen to contain beliefs that contradict known user beliefs. Also, during construction, many possibly-useful JPs are eventually ruled out because the advisor can't verify that he holds a belief the JP contains. 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 then how are these specific JPs organized and retrieved? The other is to first try the JPs that are most likely to prove useful, but then how can the most appropriate JP be selected?

Finally, we are trying to extend the model toward being a more complete dialog participant, a task that raises one other important question: How is the particular belief to justify selected? The advisor 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.

5. Comparison with Related Work

There are several classes of related systems. The first includes systems that try to provide advice to novice UNIX users. Existing UNIX advisors such as UC (Wilensky et al. 1988) and SC (Hecking et al. 1988) do not 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 "How do I remove a file?", "What does 'rm -i' do?", and so on. The other classes are systems that try to correct user misconceptions and systems that try to understand or participate in arguments.

5.1. *Explaining user misconceptions*

Our own earlier effort, AQUA (Quilici 1989a; Quilici et al. 1988; Quilici et al. 1986) tried to correct plan-oriented user misconceptions. AQUA worked by taking a single user belief (that a plan is applicable to a goal, or that a state is an enablement or effect of a plan) and used strategies that try to justify not holding the belief. A serious problem with AQUA was that it had strategies only for why people *do not* believe something, and not for why they do. It had no knowledge about how to justify a belief that a plan is preferred to another plan or has a particular state as an enablement or effect. This made it impossible for AQUA to understand the user's belief justifications, and it severely limited AQUA's ability to construct belief justifications.

ROMPER (McCoy 1989) tried to explain a different class of user misconceptions, mistaken user beliefs involving misclassifications or misattributions involving objects. ROMPER applied an approach similar to AQUA's, using a set of strategies for justifying its beliefs to the user. However, ROMPER's strategies differed in that they combined justifications for why the user might have held a belief with justifications for why the system did not. Because of this lack of separation, and because it possessed only a few strategies for only two kinds of mistaken beliefs, ROMPER suffered from the same drawbacks as AQUA.

SPIRIT (Pollack 1986a, 1986b) tried to detect and correct the mistaken plans of users of a computer mail program. Although SPIRIT's overall task was similar to ours, it took a completely different approach. Rather than trying to infer which beliefs the user was attempting to justify, it tried to infer the user beliefs underlying the user's mistaken beliefs. In addition, rather than trying to justify its own beliefs to correct user misconceptions, it simply pointed out which of the user's underlying beliefs were incorrect.

Finally, there's a large collection of tutoring systems that attempt to correct and explain user misconceptions (Sleeman and Brown 1982). 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. Moreover, because these systems simply retrieve explanations, they can handle only those misconceptions they know about in advance.

5.2. *Systems that process arguments*

Two other systems considered the problems of constructing or comprehending belief justifications in the context of participating or understanding editorial arguments.

The first, ABDUL/ILANA (Flowers et al. 1982), argues about responsibility for historical events. Its knowledge about belief justification is also in rules similar to our JPs. One rule is: X did not attack first if Y preceded X's attack with an attack act. The problem with these rules is that they are tied to specific types of events, such as an attack, that are found in just a few domains. JPs, on the other hand, are tied only to abstract planning relationships independent of a particular domain.

The other system, OpEd (Alvarado et al. 1986), recognizes plan-oriented belief justifications in economic editorials. Its primary knowledge about belief justification is in its argument units (AUs). One AU is AU-OPPOSITE-EFFECT, which says that one can believe P is bad if one's opponent believes that P achieves a goal G, but one does not believe P achieves G because one believes P thwarts G. The problem with AUs is that they're tied to knowledge about argument structure (such as the connections between the beliefs of different participants) and not limited to knowledge about belief justification (a plan not achieving a goal because it somehow thwarts the goal). This makes it difficult to use them for constructing justifications, a task not considered by OpEd.

6. Conclusions

This paper has presented a model of the process of providing corrective explanatory responses to misconceptions made by novice UNIX users. To provide these responses, the advisor must be able to formulate justifications for its own beliefs and to understand the belief justifications of the user. Both tasks require abstract knowledge about belief justification. This knowledge is represented in a collection of justification patterns that capture knowledge about possible ways to justify different types of plan-oriented beliefs. This approach differs from earlier efforts in two ways. First, its knowledge about

belief justification depends only on the abstract planning structure of the different types of beliefs being justified, and not on the specific domain (knowledge about events) or the specific task (arguing or correcting misconceptions). Second, it demonstrates how the same knowledge about belief justification can be used to build a model of the user's beliefs, and to provide an appropriate advisor response, not for just one or the other.

Justification patterns are important because they represent knowledge about an entire class of justifications. This knowledge allows novel belief justifications to be formed and understood, so long as they fall into one of the known belief classes. It also eliminates the need for large collections of specific justifications. Justification patterns potentially benefit any system, such as an intelligent user interface or tutor, that wishes to recognize and respond to the mistaken beliefs and belief justifications of its users.

7. Recent Developments

The work reported in the original version of this paper was the starting point for a considerable amount of subsequent research.

7.1. *Our subsequent research efforts*

Our initial focus was to provide a more detailed model of the process of forming advisor responses (Quilici 1989b). In particular, this work elaborated the process of providing an appropriate belief justification. It showed how justification patterns could be used to control the search for the relevant advisor beliefs to provide as a response, as well as how the advisor's response could provide a belief justification that took advantage of previously stated user beliefs (and was therefore more likely to be accepted by the user than a canned justification that didn't take these beliefs into account).

Our next step was to determine whether our model could be generalized beyond simple advice-giving dialogs in the UNIX domain. In particular, we explored how to apply our approach to a restricted class of debates about everyday planning: debates in which each dialog response presents a belief, each belief addresses some perceived disagreement, and each belief is part of a justification for some other belief related to the dialog (Quilici 1992; Quilici 1991; Quilici 1990). These efforts showed that justification patterns contributed toward addressing two major problems in discourse processing: understanding the underlying connections between successive dialog utterances and producing coherent dialog responses.

Our efforts then turned back to the UNIX advising world, and we developed a considerably more complete model of the advice-giving pro-

cess (Quilici 1994). In particular, this work showed how focus heuristics could effectively direct the search for the particular justifications that related user beliefs to beliefs provided earlier in the dialog. In addition, we showed how many advisor dialog responses could be modeled as directly addressing unstated user beliefs that were inferred from the justification patterns used to relate user responses to earlier dialog responses.

7.2. *Current open problems*

Despite all of this research effort, several important open problems remain, and we are currently working to address them.

The first is that while our justification patterns are useful, they are clearly not complete. For example, we currently can't represent responses involving class-based or example-based justifications, nor can we deal with responses involving probabilistic justifications. We also have no mechanism for representing responses that include meta-justifications: reasons for why the justifications for executing one plan outweigh the justifications for executing another plan. As a result, our representation for belief justifications needs to be extended considerably.

The second is our advisor's ignorance of the purpose of the advisory dialog. All along, we have simply modeled the advisor's role as a process of detecting and responding to incorrect user beliefs. However, users participate in a dialog for a purpose, such as to achieve particular knowledge goals (e.g., knowing an acceptable plan to achieve a domain goal). It appears as though these knowledge goals can help the advisor choose which user beliefs to respond to, as well as when to provide responses that do not simply justify a belief (e.g., providing an alternate plan for a goal). As a result, we need a more sophisticated model of the advisor's response processing that takes into account user knowledge goals.

The third is that our approach considers only those user beliefs necessary to relate the user's stated belief to the dialog. While this constrains the belief inference process, it means that the advisor's responses cannot address the underlying reasons why a user holds a stated belief (e.g., why the user does not want to be asked questions). Fortunately, justification patterns suggest a mechanism to use to infer these user beliefs; namely, trying to find instantiated JPs that can justify them. As a result, our model must be extended to determine exactly when to use justification patterns to try to infer justifications for why users hold the beliefs they state.

Finally, our advisor is currently passive: it doesn't actively seek out information from the user. However, it's reasonable for an advisor to ask the user questions as part of the advice-giving process (e.g., asking "why?" after the user specifies that answering questions is undesirable). This active

understanding can help the advisor determine the underlying reasons why the user holds beliefs that conflict with the advisor's beliefs, as well as provide other needed information that helps the advisor determine the most appropriate response. As a result, our model must be extended to include a mechanism for determining when it is desirable to ask the user questions, as well as a mechanism for understanding the user's answers.

Despite all of the open problems that must be addressed before we can have an on-line UNIX advisory system, it's encouraging that there are potential solution paths that build on our previous work. This suggests that our original paper may eventually prove to be have been an important first step.

Notes

* This paper primarily reports work done when the author was a graduate student in the Computer Science Department of the University of California, Los Angeles. It also describes later work that was supported by NSF Research Initiation Award #9309795.

1 By "explicitly provided beliefs" we refer to beliefs that the user has stated explicitly or that can be readily inferred from the user's questions. For example, if the user asks "How do I do *X*?", we can automatically infer that the user has *X* as a goal and that the user believes that there is some plan that achieves *X*.

References

Alvarado, S., Dyer, M. G. & Flowers, M. (1986). Editorial Comprehension in OpEd through Argument Units. In *Proceedings of the Sixth National Conference on Artificial Intelligence*. Philadelphia, PA.

Carberry, S. (1989). Modeling the User's Plans and Goals. In Kobsa, A. & Wahlster, W. (eds.) *User Modeling and Dialog Systems*. New York, NY: Springer Verlag.

Dyer, M. G. (1983). *In-depth Understanding: A Computer Model of Narrative Comprehension*. Cambridge, MA: MIT Press.

Flowers, M., McGuire, R. & Birnbaum, L. (1982). Adversary Arguments and the Logic of Personal Attacks. In Lehnert, W. & Ringle, M. (eds.) *Strategies for Natural Language Processing*. Hillsdale, NJ: Lawrence Erlbaum.

Hecking, M., Kemke, C., Nessen, E., Dengler, D., Gutmann, M. & Hector, G. (1988). The SINIX Consultant – A Progress Report, Technical Memo 28. University of Saarbrücken.

Kautz, H. & Allen, J. (1986). Generalized Plan Recognition. In *Proceedings of the Sixth National Conference on Artificial Intelligence*. Philadelphia, PA.

McCoy, K. (1989). Reasoning on a Highlighted User Model to Respond to Misconceptions. In Kobsa, A. & Wahlster, W. (eds.) *User Modeling and Dialog Systems*. New York, NY: Springer Verlag.

Pollack, M. (1986a). 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*. New York, NY.

Pollack, M. (1986b). Inferring Domain Plans in Question-answering, Ph.D. Thesis, Department of Computer Science, University of Pennsylvania.

Quilici, A., Dyer, M. G. & Flowers, M. (1986). AQUA: An Intelligent UNIX Advisor. In *Proceedings of the Seventh European Conference on Artificial Intelligence*. Brighton, England.

Quilici, A., Dyer, M. G. & Flowers, M. (1988). Recognizing and Responding to Plan-oriented Misconceptions. *Computational Linguistics* **14**(3): 38–51.

Quilici, A. (1989a). AQUA: A System that Detects and Responds to User Misconceptions In Kobsa, A. & Wahlster, W. (eds.) *User Modeling and Dialog Systems*. New York, NY: Springer Verlag.

Quilici, A. (1989b). The Correction Machine: Formulating Explanations for User Misconceptions. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*. Detroit, MI.

Quilici, A. (1990). Participating in Plan-oriented Dialogs. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society*. Boston, MA.

Quilici, A. (1991). The Correction Machine: A Computer Model of Recognizing and Producing Belief Justifications in Argumentative Dialogs. Ph.D. Thesis, Department of Computer Science, University of California.

Quilici, A. (1992). Arguing about Plan Alternatives. In *Proceedings of the 17th Annual Meeting of the Computational Linguistics Society*. Nantes, France.

Quilici, A. (1994). Forming User Models by Understanding User Feedback. *User Modeling and User Adapted Interaction* 3(4): 321–358.

Sleeman, D. & Brown, J. S. (eds.) (1982). *Intelligent Tutoring Systems*. Orlando, FL: Academic Press.

Wilensky, R., Chin, D., Luria, M., Martin, J., Mayfield, J. & Wu, D. (1988). The Berkeley UNIX Consultant Project. In *Computational Linguistics* **14**(4): 35–84.

Wilensky, R. (1983). *Planning and Understanding*. Reading, MA: Addison Wesley.