

Utilizing User Models to Handle Ambiguity and Misconceptions in Robust Plan Recognition

RANDALL J. CALISTRÌ-YEH

*ORA Corporation
301A Harris B. Dates Dr.
Ithaca, NY 14850-1313
calistri@oracorp.com*

(Received 14 September 1990; in final form 1 February 1991)

Abstract. User models play a critical role in robust plan recognition by controlling ambiguity. They allow an observer to prefer one plan explanation over another, and they provide a means of measuring the believability of misconceptions when the user's plan is flawed. This paper discusses the motivation for including a user model in robust plan recognition, and shows how a probabilistic interpretation offers a practical means of incorporating a user model into the plan recognition process.

Key words: user modeling, plan recognition, probabilities, misconceptions

1. Introduction

Plan recognition is essentially based on a user model; all but the most trivial forms of plan recognition require previously drawn assumptions about the user, and as the plan recognition task becomes more robust and more complicated, the reliance on a model of the user increases. This becomes even more apparent when both plan recognition and user modeling are viewed in the context of intelligent interfaces such as those of Calistrì-Yeh (1990), Carberry (1990), Cheikes (1989), Goodman and Litman (1990, 1991), and Sider and Burger (1991). When a computer system is expected to interact intelligently with a user, the job of recognizing the user's plans is crucial, and that job relies heavily on information about the user. An important part of the plan recognition process for an intelligent interface is the ability to handle cases where the user's plan is flawed in some way.

We have developed an approach to robust plan recognition that can handle situations where the user's plan contains misconceptions. We have implemented a program called Pathfinder, which uses a new classification of plan-based misconceptions to extend a plan recognition algorithm based on A* to handle faulty plans. The search through the plan hierarchy is guided by

probabilities supplied by the user model to allow more likely explanations to be preferred over less likely ones.

We feel that there are several advantages to our approach. The classification that we present for plan-based misconceptions is complete (given certain reasonable assumptions about the structure of plans), and covers more types of misconceptions than other proposed classifications. It is also domain-independent, and has been successfully tested in several different domains. Since the classification is directly related to plan structure, it can be easily incorporated into the plan recognition algorithm. Finally, since the Pathfinder program is guided by probabilities, we are able to quantitatively prefer one plan explanation over another, and we can express a degree of confidence in our selection.

We shall begin by discussing the features that are needed in a user model to support plan recognition. We will then describe how a probabilistic user model can be used with “traditional” plan recognition, and how it can be extended to deal with robust plan recognition involving misconceptions. Next, we will present our classification of plan-based misconceptions and our Pathfinder program. We will describe how the likelihood of a misconception can be decomposed into a number of “features”, and we will provide details of how the user model is realized in the Pathfinder program. Finally, we will analyze the performance of the classification, the plan recognition algorithm, and the user model when tested on a large corpus of naturally occurring examples of plan-based misconceptions.

2. User models and plan recognition

Kass and Finin (1988) have proposed several different dimensions over which user models can be categorized. One common way is to use stereotypical models of “expert” and “novice” users to tailor a response to the correct level of expertise (Paris, 1985). Paris suggests that this can be generalized to allow a user to be an “expert” in certain areas and a “novice” in others (Paris, 1988). Other researchers, such as McLoughlin (1987) and Rich (1979) allow the modeling process to include a larger set of stereotypes. Kass attempts to tailor a model for each specific user (Kass and Finin, 1988).

There is also a need for user modeling that specifically relates to plan recognition. Virtually all interesting cases of plan recognition in realistic domains involve some amount of ambiguity. Goldman and Charniak point out that as an observer traces the first few actions of an agent, or as a listener hears the first few sentences of an utterance, the complete plan that is being followed is often ambiguous (Goldman and Charniak, 1988).

For example, if we were to hear the sentence “Jack went to the supermarket”, we could come up with many possible plans that Jack could be following: he could be going there to do some shopping, he could be going there to meet someone, he could be going there to rob the store, or he may just be out for a stroll. In most cases, however, the observer or listener would not treat this as an ambiguous utterance, certainly not as ambiguous as it has just been portrayed. Most people would immediately jump to the “obvious” conclusion that Jack is going to the supermarket to do his shopping, and would not even consider the hundreds of other possible explanations. But if we think about *why* we consider this to be the obvious plan, we would probably justify our choice with the explanation that this is the “most likely” plan that he would do under “most” circumstances. In other words, this is what our model of Jack tells us that he would do. Even if we had never met Jack before in our lives, we would still strongly prefer this explanation because it is what a “normal” person would do; i.e., it is the explanation that the default generic model supports.

To illustrate a case where user models are even more critical, consider the so-called “Brown Batman Problem” (Carroll, 1988). Brown University has a probabilistic natural language plan recognition system (Wimp2) (Goldman and Charniak, 1988) that handles simple stories about Jack going to the supermarket. The problem is what to do if, instead of “Jack went to the supermarket”, we encounter “Batman went to the supermarket”. Obviously, the probabilities for the various explanations need to be adjusted. The fact that the agent is Batman instead of Jack seems to make a mundane supermarket-shopping plan less likely. But at the same time, typical plans for Batman (such as chasing arch-villains) doesn’t seem quite right in a supermarket setting (after all, even Batman needs to eat). The critical point is that we can’t even address this problem without some concept of a user model: without a user model, we would only have the concept of an agent going to the supermarket, with no way to distinguish between the different goals that Jack or Batman might have.

3. What can we expect from the user model?

In its most ideal form, a user model is a complete specification of what the user will do in any situation. Given a state of the world and a stimulus, it should be able to predict with certainty exactly how the user will react to the stimulus. Such a user model would completely solve the ambiguity problem of plan recognition, since it would always tell us which plan the user would be executing in the given situation. Of course, there is no hope of acquiring the knowledge necessary for a perfect and complete user model. People don’t

even know themselves that well: everyone has had the experience of not knowing what they would do in a certain situation, or of being surprised that they reacted to something in an unexpected way. If we don't even have perfect models of ourselves, we certainly cannot expect to have perfect models of other agents.

At the opposite end of the spectrum, some "minimal" form of user model is completely unavoidable; at the very least, the system always has an implicit model of a default user. These models are formed from various assumptions that the system designers made about the types of people that will use the system (Kass and Finin, 1988), and might include assumptions about the types of plans the users may have, the beliefs they may have about the world, their preferred methods of communication, their vocabulary, and their level of expertise in the domain. Very often, these assumptions are not explicitly stated; they might only show up as design decisions during the construction of the system, and the designers might not even have made them consciously. But even though they are not normally considered "real" user models, they still fulfill the role of a rudimentary user model when an explicit model is not present.

This type of user model is undesirable for a number of reasons. First, because it is not constructed consciously, it is not planned or organized. Because of this, the user model may be scattered throughout the system, and it is often not even clear what belongs to the model and what does not. This makes it extremely difficult to modify the model, and makes it virtually impossible to reason about it directly. Also, implicit models almost invariably constitute a single model of a "generic" user, with no differentiation between different classes of people, so that any information that can be extracted from the model is likely to be inaccurate for different users. So in addition to being hard to understand and hard to modify, implicit models are also not terribly useful.

So what can we reasonably expect from a user model? To avoid the problems described in the previous paragraph, an explicit user model is necessary. We should be able to reason directly about the model, and we should be able to isolate it from the rest of the system in order to adapt it (or replace it entirely) to accommodate other users. Finally, in order to handle the task of plan recognition, it should be able to predict and describe the user's plans, goals, and actions accurately. As we shall discuss in Section 6.2, the user model should explicitly represent the likelihood of the user selecting particular plans, and should represent various "features" of potential misconceptions that the user may have (as described in Section 6.1).

4. Probabilistic user models

Since we are unable to perfectly predict the user's actions, and since plans are ambiguous, the best we can hope for is the ability to express a preference for expecting one action instead of another, and to assign some measure of confidence to this expectation. This argues for a probabilistic interpretation of user modeling, since we intuitively want the "most probable" action or explanation.

4.1. TRADITIONAL PLAN RECOGNITION

From a plan-recognition perspective, the problem we want to be able to solve is:

Given a closed world of correct plans, knowledge about the current and past states of the world, and information about a queried step in a plan (possibly described at multiple levels of abstraction), determine how this step relates to the higher level plans and goals of the agent.¹

The simplest model, from a plan recognition perspective, is to have a fixed set of probabilities for each plan in the domain, describing how likely it is for users to select this particular plan. More specifically, plans can be stored in a plan hierarchy that forms a directed *AND/OR* graph. The leaves of the graph correspond to atomic actions. *AND* nodes form collections of **substeps** ("you need to do *A and B and C*"). *OR* nodes form collections of **alt steps** ("you can either do it this way *or* that way"). Each arc of the graph can then have a probability associated with it, representing the likelihood of a user selecting the child given that he has selected the parent. Substep arcs will all have probabilities of 1.0, since a substep must be executed once the parent plan is chosen. Each alt arc will have a probability reflecting the relative likelihood of the user selecting that particular alternative.

A fragment of a sample plan hierarchy related to filling out income taxes for graduate fellowships is shown in Figure 1. Inherited parameters are shown on the arcs (i.e., when the *get-obj* plan is invoked from *fill-in-wages*, the *obj* parameter must be a *W2* form). Probabilities for alt steps are shown in parentheses (i.e., given that the user is filling in his taxes, he is 60% likely to use a 1040 form).

This type of formalism meets all of the criteria discussed above. We can isolate the user model in this representation, and the model can be altered simply by substituting a new set of probabilities. We can reason directly about

¹ Depending on the "level of abstraction", we may already know the agent's top-level goal, in which case Appelt and Pollack would consider this "plan evaluation" instead of "plan recognition" (Appelt and Pollack, 1991).

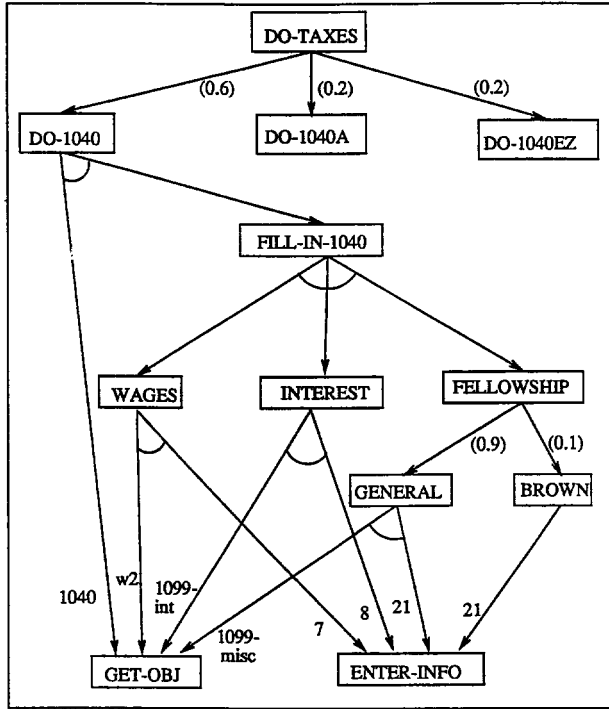


Fig. 1. A sample plan hierarchy.

the model by performing computations with the probabilities, and these values give us a means of preferring one plan over another with a measurable degree of confidence.

Notice that this does not specify whether the probabilities are for all users, for a stereotype class of users, or for one specific user. While the implicit model was constrained to only model a generic user, we have much more flexibility with the explicit model. Since the model itself can be completely isolated and replaced, we could conceivably have a separate model tailored to each individual user, and when the system is reasoning about a particular user's actions, it will use the appropriate model. Similarly, the system could be provided with a function that maps users onto a set of stereotype classes, again with the appropriate model being retrieved for each user. All we need to be able to do is capture the fact that certain people may be more likely than others to choose a particular plan.

In most cases, simple fixed probabilities will not be sufficient to define the likelihood of a particular course of action. For instance, the most probable plan that an agent will choose for investing his money might be strongly

dependent on factors such as the prevailing interest rates and the stability of the stock market. We need to capture the context sensitivity of the user model, so single probabilities on the arcs should be replaced by probability distributions which are able to take other facts about the current state of the world into account.

This type of probabilistic user model gives us the ability to allow multiple interpretations of ambiguous situations while still expressing a preference for the “most likely” explanation of a user’s actions.

4.2. ROBUST PLAN RECOGNITION

Unfortunately, life is not as simple as this. A tacit assumption in this view of plan recognition (one that has been made all too often) is that the user never makes any mistakes. But people do make mistakes, and plan recognition systems should be robust enough to handle circumstances where the user’s actions do not match any known correct plan. We refer to this situation as “robust plan recognition”, to distinguish it from the “perfect plan recognition” problem presented earlier:

Given a closed world of correct plans, knowledge about the current and past states of the world, and information about a queried step in a plan (possibly described at multiple levels of abstraction), determine how this step relates to the higher level plans and goals of the agent, assuming that the agent’s plans may be flawed in unpredictable ways.

The phrase “unpredictable ways” does not mean that incorrect plans are incapable of being classified into predictable categories. Rather, it means that it is not possible to predict all specific incorrect plans ahead of time.

Once we allow the possibility that the user may have flawed plans, i.e. that he may have misconceptions about how his plans are constructed or about how they affect the world, the plan recognition process is much more difficult, and user models become even more critical. Although the problem of ambiguity already exists in perfect plan recognition, flawed plans are *inherently* ambiguous (and ambiguous to a much higher degree), because there are many potential flawed plans that could be attributed to a user to explain a discrepancy between his actions and known correct plans. The system must have some way of selecting among all of these competing explanations for the actions or utterances of the user, and this information must come from the model of the user that the system has.

One way of incorporating this information into the user model is to use a method similar to the mechanism for ranking correct plans: have a set of probabilities for each flawed plan in the domain, describing how likely it is for users to select this particular plan. This is commonly known as the “bug

list” approach (Brown and Burton, 1978; Anderson et al., 1985). In order to use this approach, it is necessary to assume that there is a relatively small number of known mistakes that the user can make. In other words, the user is allowed to make mistakes, but only predictable mistakes that have been pre-coded into the system. This is a useful and efficient way to deal with a few of the most likely mistakes, and it allows the flawed plans to be handled exactly like the correct plans. But this method is not sufficient by itself; it places a great strain on the user model to anticipate all of the likely mistakes, and bug lists do not take a realistic view of the impressive creativity that people exhibit in their flawed plans. People are capable of an infinite variety of misconceptions, and it is quite unlikely that the plan knowledge base will have a pre-stored plan for the specific mistake that a user will make.

The alternative that we present here does not attempt to anticipate all of the possible invalid plans at all. Instead, it “creates” the invalid plans by dynamically modifying and annotating a known correct plan to match the current situation. The problem of ambiguity, which is even greater in flawed plans than it is in correct plans, can still be dealt with by using a probabilistic user model. But since this method does not explicitly store all of the possible invalid plans, the user model cannot assign probabilities directly to an invalid plan as it did with valid plans. Instead, it assigns a probability to the plan-based misconception that caused the plan to be invalid.

Why do we need to assign probabilities to misconceptions in the first place? Just as different types of users will choose different types of plans, they will also make different types of mistakes. A professional tax consultant will not make the same kinds of mistakes on his income tax form as a high school student. Similarly, native speakers of Spanish will make different mistakes in translating from Spanish to English than native speakers of English (Schuster, 1985). Such tendencies toward certain types of mistakes can be captured in the user model, similar to the way that the model captures tendencies to prefer certain plans over others.

It may be critical to diagnose the mistake correctly, because attempting to correct a misconception that the user doesn’t have will only serve to confuse him more. For instance, consider the following query:

“I’m trying to fill out my taxes. Where do I get the W2 form for my fellowship?”

It turns out that W2 forms are only used for wages, and that fellowships normally get 1099 forms instead. But in this particular case, the user’s fellowship is from Brown University, and so does not get any form at all. There are several possible explanations for the user’s mistake:

- he might think that fellowships are really a form of wages (incorrect), which get W2 forms (correct);

- he might think that fellowships are separate from wages (correct), but that fellowships do get W2 forms (incorrect);
- he might think that wages get W2 forms (correct), and that most fellowships get 1099 forms (correct), but that Brown fellowships are special (correct) and get W2 forms (incorrect).

Trying to correct the wrong problem could be disastrous: for instance, the simple response “You don’t need a form.” would be appropriate in the third case, but might end up giving the user additional misconceptions in the first two cases.

So it is important to be able to judge the relative likelihood of mistakes. Unfortunately, we have the same problem with the plan-based misconceptions that we had with the invalid plans themselves; namely, that it is not possible to pre-assign a probability to each possible misconception because there are an infinite number of mistakes that a user could conceivably make. It is necessary to find some way of preferring one misconception to another, and of preferring one plan (valid or invalid) to another, that does not require knowing the misconception ahead of time.

In order to handle these cases, we have developed a classification of plan-based misconceptions, along with a robust plan recognition algorithm that uses this classification to assign relative probabilities to the various possible explanations for a user’s query, even when that query contains misconceptions.

5. The Pathfinder system

Before discussing how a user model can help with this problem of robust plan recognition, we first need to describe the various types of plan-based misconceptions, and we need to discuss the algorithm used to perform robust plan recognition.

5.1. CLASSIFYING PLAN-BASED MISCONCEPTIONS

In order to handle flawed plans, we have developed a complete classification of plan-based misconceptions, covering all ways that plans with a certain structure can fail (Calistri, 1990, Ch. 4). There are ten detectable and distinguishable classes of plan-based misconceptions, which are derived directly from the structure of plans².

² The original classification contained 16 categories, several of which were either collapsed together because they could not be distinguished by an outside observer, or were eliminated because they were undetectable. The details of the simplification to 10 categories are not important for this paper, but can be found in (Calistri, 1990, Ch. 4).

Since we are taking a structure-based approach to misconception classification, we will briefly describe the various components that make up the structure of a plan. The plans that are considered here have the following four components:

- **Parameters** are placeholders for objects that are used by the plan.
- **Preconditions** are propositions about the world that must be true for the plan to be executed. Preconditions are always beyond the control of the agent; if he *can* control a condition, it will be represented as a **substep**, since he must have a plan for achieving it.
- **Steps** are things that need to be done in order to satisfy a plan. There are three different types of steps:
 - = **Substeps** are necessary and sufficient steps that must be carried out in order to do the plan.
 - = **Alts** are alternate ways of doing a plan.
 - = **Add and delete** are lists of propositions that are used to model atomic actions.
- **Temporal constraints** place certain restrictions on the timing of steps. There are two different types of temporal constraints:
 - = **Orderings** are temporal constraints that enforce a partial order on substeps.
 - = **Mutex** constraints partition the alt steps into subsets of mutually exclusive steps.

Plans are grouped together in a hierarchy that forms a directed *AND/OR* graph, as illustrated in Figure 1.

We now present the ten classes of plan-based misconceptions, illustrated with examples from the plan hierarchy in Figure 1. A discussion of the completeness of this classification is beyond the scope of this paper; interested readers are referred to (Calistri, 1990).

1. **Violated Binding (VB):** Certain plan parameters can be restricted by the calling plan. If the agent attempts to use an incompatible binding, a violation occurs. For example, the `get-obj` plan has a parameter for the object that the agent wants to get. When this plan is invoked as a substep of the `fill-in-general-fellowship` plan, this parameter will automatically be bound to a 1099 form. If the agent tries to get a W2 form instead, it will violate the binding that already exists.
2. **Violated Precondition (VP):** Since a precondition is something that must be true for a step to be executed, a precondition whose value is false when the agent tries to execute the step is an obvious failure. For example, in order for the agent to get a 1099 form for his fellowship, the fellowship must not be from Brown University. If his fellowship is from Brown, then the step will not succeed.

3. **Substituted Precondition (SP):** In this case, the agent is aware of the precondition, but his version of the precondition is incorrect. For example, he may think that he can get a 1099 form as long as his fellowship is not from Cornell (instead of Brown).
4. **Missing Precondition (MP):** Here, the agent is not aware that the satisfaction of this particular precondition is necessary for the success of his plan. The difference between missing and violated preconditions is a bit subtle. In the case of a missing precondition, the agent does not even know that the precondition exists (“I didn’t know that my fellowship couldn’t come from Brown”). With a violated precondition, the agent knows about the precondition, but is presumably unaware that it has been violated (“I know Brown doesn’t give out 1099 forms — I thought my fellowship was from the government”).
5. **Extra Precondition (EP):** Here, the agent thinks that the satisfaction of this precondition is necessary for the success of his plan, but in fact it is not necessary. Continuing the previous example, the agent might think that he also has to be a first-year graduate student in order to get a 1099 form.
6. **Violated Optimization (OPT):** It may be the case that two branches of an alt node are both “correct” in the current situation, in the sense that both are valid plans and that there are no actual violations. But one of the alternatives may be preferred over the other. If the agent chooses a sub-optimal path, then there is a **violated optimization**. An example of this might be if the agent has a large number of deductions but chooses not to itemize. He is not required to itemize, but if he did he would get a larger refund.
7. **Substituted Step (SS):** The agent is attempting an incorrect step in place of the correct step. An example of this would be if the agent tried to record the money from his fellowship by using *fill-in-wages* instead of *fill-in-fellowship*.
8. **Missing Step (MS):** The agent should have included this step in his plan, but didn’t. For example, he might not know that he has to fill in his fellowship information when he completes his 1040 form.
9. **Extra Step (ES):** The agent is including a step that doesn’t belong. These are exactly analogous to missing steps.
10. **Violated Ordering (VO):** The agent thinks (correctly) that step *A* has to be done before step *B*, but thinks (incorrectly) that step *A* has already been done. There are also substituted, missing, and extra orderings, but to an outside observer they are indistinguishable from violated orderings.

In addition to our completeness claim, a major advantage of our classification is that since it is based on the structure of plans, it is easy to incorporate

the classification directly into a plan recognition algorithm. This is in contrast with other classifications that are based on complex reasoning about belief.

This classification only addresses the user's "top-level" misconceptions (intuitively, the incorrect beliefs that directly caused his plan to fail). Other researchers, most notably Quilici, Dyer, and Flowers (Quilici et al., 1988), continue where our classification leaves off: they take a high-level plan-based misconception and attempt to find the underlying incorrect beliefs that caused it, collecting this information to generate a more helpful response.

However, these underlying incorrect beliefs may be difficult to uncover, and they might have been caused by even deeper incorrect beliefs. There is no obvious limit to the depth of this chain of mistakes, and the further down this chain we go, the deeper into the agent's mind we must travel. Since the only data an outside observer has is a surface utterance, these "deep" misconceptions are often impossible to detect.

Of course, we might know something about the particular user or about the particular situation that would lead us to prefer one of these beliefs over the others, and this would allow us to directly address his underlying misconception as well as the surface misconception that caused the plan to fail. In fact, this is exactly what Quilici attempts to do. When this is approach is possible, the observer is often able to generate a more accurate and helpful response. But we cannot always rely on having such detailed information all the time. The plan-based misconception is the only information that is needed to correct the immediate problem with the user's plan, and in many cases this information will also be sufficient to allow the user to correct his underlying incorrect belief himself.

Pollack (1986) suggests three classes of plan-based misconceptions. **Un-executable acts** occur when the agent has a false belief about the executability of a plan. This would include anything that causes an otherwise correct plan to become invalid, and encompasses the categories of violated, substituted, and missing preconditions. **Ill-formed plans** occur when the agent has a false belief about the hierarchical relationship between two steps. Since Pollack's plans merge bindings and step names, this category includes special cases of violated bindings, substituted steps, missing steps, and extra steps. These are just special cases because she only considers "simple" plans which consist exclusively of simultaneous substeps. Finally, **incoherent queries** occur when the observer cannot find any way of attributing a rational plan to the agent. Pollack's canonical example of an incoherent query is

"I want to call Kathy at the hospital. How do I stand on my head?"

Here, the expert has absolutely no idea why the user thinks that standing on his head will help in his plan to call Kathy. The classification developed in this chapter has nothing that corresponds precisely to an incoherent query. It

would treat this case as a particularly bad extra step. Clearly the user believes that standing on his head is part of a phoning plan, and since it does not correspond to any known steps for this plan, it must be an extra step. The reason why Pollack creates a separate category for examples like these is that she does not have any way of measuring the likelihood of a misconception, and she wants to distinguish these cases from cases of more “believable” extra steps. We believe that it is more appropriate to handle this with probabilities.

Quilici et al. (1988) presents three slightly different categories of plan-based misconceptions. **Violated plan applicability** implies that a particular plan should not be used to achieve a goal. It roughly corresponds to a substituted or extra alt step. **Violated enablement** occurs when the agent incorrectly believes that a particular state exists (or must exist) before a plan can achieve a goal, and covers the four types of precondition misconceptions. And **violated effects** occur when the agent has incorrect beliefs about the state that will exist as a result of the plan’s execution. Since the plan structure in Section 5.1 only represents effects at the atomic level (via the add/delete lists), my classification can only address special cases of violated effects.

Van Beek (1987) uses a classification similar to that presented by Joshi et al. (1984b). This classification focuses on whether the query is possible, whether it achieves the intended goal, and whether there are other alternate ways of achieving the goal. The five relevant combinations that produce plan-based misconceptions are:

1. Failed query, another alternative.
2. Failed query, no alternative.
3. Successful query, does not achieve goal, another alternative.
4. Successful query, does not achieve goal, no alternative.
5. Successful query, achieves goal, better alternative.

He does not distinguish between the various ways that the query can fail, and treats steps and preconditions interchangeably. So his “failed query” matches all of the precondition and step misconceptions (except extra precondition, which he does not address). My classification does not differentiate between whether there are other possible alternatives (except for violated optimizations, which match his last category). His classification does not deal with bindings or orderings at all, and does not handle missing steps because of the type of query that he assumes.

Figure 2 summarizes how these three classifications map onto the classification presented in this chapter. The columns represent the ten types of plan-based misconceptions from my classification, along with Pollack’s “incoherent plan” and Quilici’s “violated effect”. This figure illustrates that our classification encompasses a superset of the other three systems. The one type of misconception that it does not represent explicitly is “incoherent

	1	2	3	4	5	6	7	8	9	10	11	12
Calistri-Yeh	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	n	s
Pollack	c			n	s				n	n	Y	n
Quilici	c				c			n	n	n	n	Y
vanBeek	c			n	s			n	n	Y	n	n

1. Subst. Precondition	7. Extra Step
2. Violated Precondition	8. Missing Step
3. Missing Precondition	9. Violated Ordering
4. Extra Precondition	10. Optimization
5. Violated Binding	(11. Incoherent Plan)
6. Substituted Step	(12. Violated Effect)
"Y" = classification handles this type "c" = classification collapses several types "s" = classification only handles special cases "n" = classification does not handle this type	

Fig. 2. Comparison of the four classifications.

plan”, which is indirectly addressed by probability measures on extra steps. It is quite significant that we can handle categories that are not addressed at all by the other three classifications; it may be less significant that several of our categories are combined into a single category in other classifications. It is still unclear whether this more fine-grained representation of plan-based misconceptions is important for cooperative response generation. It may be that such specific categories are critical for correcting the user’s problems, or it may be that several categories will always be treated the same in a response, in which case a coarser representation is acceptable. The only way to determine this is by more experimentation with actual response generators.

5.2. DETECTING PLAN-BASED MISCONCEPTIONS

We have developed a program called Pathfinder that does robust plan recognition in the presence of plan-based misconceptions. It uses a probabilistic best-first search method based on the A* algorithm (Hart et al., 1968; Slagle and Bursky, 1968), as described in Pearl (1984, Ch. 3.1) to search a plan hierarchy for the most likely explanation of an agent’s action (Calistri-Yeh, 1991). Probabilistic approaches to plan recognition (or, in general, methods

that allow one explanation to be quantitatively preferred over another) are fairly new. Some other notable approaches are Carberry (1990), Goldman and Charniak (1988, 1989), Appelt and Pollack (1991), and Raskutti and Zukerman (1991). However, except for Appelt and Pollack, none of these systems address the possibility that the user may have a faulty plan.

Appelt and Pollack's "weighted abduction" approach has certain similarities to our probabilistic methods, even though their system is heavily belief-based while ours is structure-based. They tend to gloss over the issue of "correct" vs "incorrect" beliefs, instead requiring only that the user's collection of beliefs is internally consistent. Incorrect beliefs are treated just like correct beliefs, except that there is usually a higher "penalty" weight for assuming the belief. This is similar to the way that we typically assign lower probabilities to misconceptions than to correct plans.

The Pathfinder algorithm uses two different methods to handle misconceptions: one for **structural** misconceptions, and one for **constraint** misconceptions. Plans that contain structural misconceptions will be referred to as **ill-formed**, and include cases of substituted, missing, and extra steps. Plans that contain constraint misconceptions will be referred to as **violated**, and encompass violated bindings, temporal violations, and misconceptions involving preconditions. Plans which have no misconceptions at all are called **valid**.

The crucial difference between structural and constraint misconceptions is the way in which the misconception is reflected in the plan hierarchy. For constraint misconceptions, the connections between nodes in the graph are not affected. A path which has only constraint misconceptions can still be located directly in the plan hierarchy. Paths with structural misconceptions, on the other hand, do not exist in the correct plan hierarchy at all. For example, representing an extra step involves adding a new arc to the hierarchy. In order to recognize ill-formed plans, the actual topology of the graph must be altered.

The algorithm is shown in Figure 3. It works by incrementally extending partial solution paths until one of them reaches the destination node. It maintains two lists: an OPEN list, which holds the partial paths that are being considered for expansion, and a CLOSED list, which holds partial paths that have already been expanded and might be used in the solution path. We start with a single partial path consisting of the user's top-level goal, and place it on the OPEN list. We then expand this partial path by generating all ways that it can be extended to the first mentioned node (these new path segments are called "successors" of the original path).

For each of these successors, we verify every step in the new partial path by checking to see whether any constraints (preconditions, temporal orderings, bindings) have been violated. If there are any violations, we flag them

1. OPEN \leftarrow goal
CLOSED \leftarrow \emptyset .
2. If OPEN = \emptyset then exit with failure.
3. Remove partial path p with optimal $f(p)$ from OPEN and place on CLOSED.
4. If the endpoint of p is the queried step, exit with the complete path formed by tracing the pointers back to goal.
5. Otherwise, **Expand**(p):
 - 5.1. Generate all successors of p (all partial paths from the endpoint of p to the next known node, plus other ill-formed partial paths), with pointers back to p . If a successor is ill-formed, add the appropriate documentation to the path.
 - 5.2. For each successor p' of p , **Verify**(p'):
 - 5.2.1. Check p' for violations.
 - 5.2.2. If p' contains any violations, add the appropriate documentation to the path.
 - 5.2.3. Regardless of violations, estimate $f(p')$ and add p' to OPEN.
6. Go to step 2.

Fig. 3. The Pathfinder algorithm.

as misconceptions. We then compute a heuristic function f . This function estimates our confidence that this partial path will be part of the final answer, and takes into account the believability of any misconceptions. Each of the verified successors is then added to the OPEN list. Then we remove the original path, which has already been expanded, and place it on CLOSED, where we can retrieve it if we need it for the final answer.

Finally, we go back to the OPEN list and choose the most likely partial path for expansion, based on the heuristic function that is stored with the partial path. We continue to repeat the entire process until we find a partial path that reaches the specific query that the user made.

We handle ill-formed plans by modifying the method of generating successors to include additional partial paths that do not actually exist in the plan hierarchy. These paths could result from substituting one step for another, or from adding a step to a path that does not contain it. They will involve changing the structure of the plan hierarchy graph by adding or deleting nodes or arcs in order to form new paths. The process is controlled by only proposing

“reasonable” modifications based on the misconception features described below. Pathfinder generates these modifications in such a way that at least one path (explanation) is always guaranteed to be found, so the failure exit in step 2 of the algorithm will never be reached.

Pathfinder can recognize plans with any number of misconceptions. The multiple misconceptions can be closely related, or can be from completely separate classes. In one real-life example from the transcripts described in Section 7.1, Pathfinder successfully recognized a plan containing three distinct misconceptions (a missing precondition, a violated binding, and a violated precondition). Because of the way the heuristic function f works, Pathfinder is able to take the seriousness of the misconception into account, and can handle cases where several “believable” misconceptions are more likely than a single unlikely misconception. In fact, it can even handle cases where a reasonable flawed plan is more likely than a very obscure perfect plan.

The user model comes into play during the computation of f : Pathfinder’s search is controlled by this heuristic function, which in turn is calculated from the probabilities found in the user model. There are two types of probabilities that are needed: the probability of selecting an alternative given the parent node (already described in Section 4.1), and the probability of having a particular misconception given a particular plan.

6. Handling misconceptions with a user model

Now that we understand a bit about the method for recognizing misconceptions and the role that the user model plays in this method, we can return to our discussion of how to actually calculate the probability of a user having a particular misconception.

Since Pathfinder constructs misconception nodes dynamically, we need to find a way to generate the probability of a misconception “on the fly”. The approach taken here is to generate these probabilities based on the classification of the misconception. In other words, each category in the classification will have a formula for calculating the probability of any misconception belonging to that category. Using that formula on the particular misconception will produce the desired probability.

6.1. MISCONCEPTION FEATURES

It is possible to break the probability of a misconception down into a number of **features**. These features can be calculated independently from readily available information, and can then be combined to form the probability

itself³. So far we have isolated six major features of plan-based misconceptions. This is not meant to be an exhaustive list by any means, and there are many other subtle factors that can influence a misconception. But studies of a large set of transcripts and experiments with Pathfinder have suggested that these six features provide a surprisingly reliable estimate of the likelihood of the misconception (see Section 7).

The six features that make up the probability of a misconception are **similarity**, **obscurity**, **complexity**, **overgeneralization**, **permanence**, and **discourse**.

- **Similarity** compares the agent's incorrect plan component with the correct version of the component to determine how similar they are. This feature is especially relevant for violated and substituted components, since it can measure how close the agent's version is to the correct version. It is irrelevant for missing and extra components, because these types of misconceptions do not have both a correct and incorrect version that can be compared.
- **Obscurity** measures how familiar the component is expected to be to the agent. People are more likely to have misconceptions about uncommon, little-known components than about very common components. Obscurity can often be used to distinguish whether a failed precondition is violated (the user thinks he has successfully satisfied it) or missing (the user is unaware of the precondition). If the precondition has a high obscurity score, then that would favor the interpretation that the agent has a missing precondition. If the obscurity score is lower, then it is more likely that it is violated.
- The factor of **complexity** takes into account how complicated the particular component is. Generally, the more complex a component is, the more likely it is that an agent may have an incorrect version of the component. This is especially important for preconditions and steps. If a step is very complex and has a large number of complicated substeps, it is more likely that the agent may be missing certain steps (or may have the steps in the wrong order, leading to a violated ordering).
- Certain misconceptions are caused by the fact that the agent has **overgeneralized** on a particular aspect of the plan hierarchy. This is quite useful in explaining extra steps and extra preconditions. Overgeneralization can also occur in situations where there are many similar plans but one of the plans has a special case that is slightly different. This may cause the agent to have a substituted step or precondition for the special case.

³ It should be noted that the features themselves are not considered probabilities. They are simply numbers that can be combined to form a probability.

- In addition to overgeneralizing on the structure of a plan, an agent may also overgeneralize on the temporal **permanence** of a plan. For instance, a precondition for a plan may have held in the recent past, but has just changed a few moments ago. If the agent is not aware of this change, then he is very likely to have a violated precondition, believing that the precondition still holds when in fact it doesn't.
- Often, things that were said earlier in a **discourse** can influence the believability of misconceptions. For example, the observer might have already discovered that the agent has certain misconceptions about a plan. This could indicate that the agent is not very knowledgeable about this plan, which might make additional misconceptions more believable. It could also be that the observer has already diagnosed a similar (or even identical) misconception earlier in the dialogue. In this case, it is not quite as clear how this should affect the probability of the current misconception. One interpretation is that since we have already seen the mistake before, we are likely to see it again. But another possibility is that “only a fool makes the same mistake twice”; since the expert has already corrected the earlier mistake, we would not expect it to occur again.

The six features are not equally important for all types of misconceptions. Each category of misconception stresses different features, and in fact there are some misconception categories that do not have certain features at all. This is handled by associating a vector with each misconception category. This vector defines how the probability of the misconception should be constructed, by assigning weights to each feature. In violated preconditions, for example, the similarity feature is considered ten times more important than the feature of complexity.

6.2. DEFINING THE USER MODEL

Pathfinder's user model is defined by the local probabilities associated with each alternative step in the plan hierarchy (for instance, the probability of going to the supermarket versus going to the corner store, once the user has decided to go shopping), together with the information needed to calculate the misconception features and construct the probabilities of a misconception. This information makes use of three knowledge bases, which are structured as weighted discrimination nets.

There is a net for objects, which we will call the O-Net. A fragment of a sample O-Net for the income-tax example is shown in figure 4⁴. This is

⁴ The numbers shown in the sample networks are primarily for illustration, and do not reflect any deep theory.

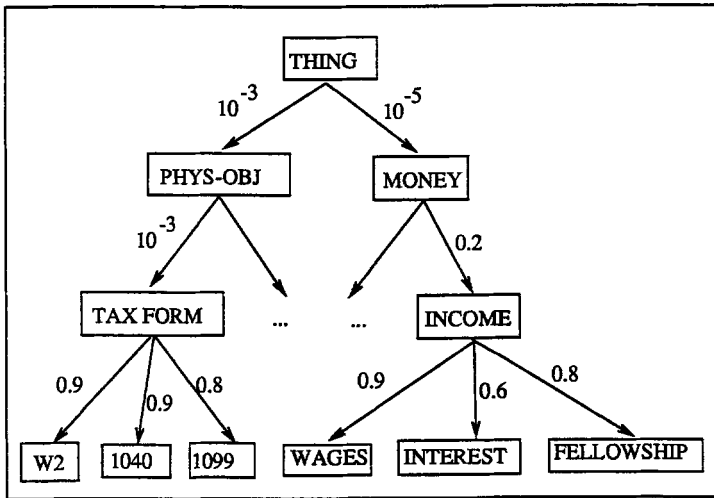


Fig. 4. Sample O-Net.

basically an *isa* hierarchy with weights on the arcs that measure how closely two objects are related. There is a corresponding net for steps, called the S-Net, illustrated in Figure 5. Any steps can be grouped together in the S-Net, regardless of their positions in the plan hierarchy, by forming new categories such as <fill-in-job-money>. The S-Net loosely parallels the plan hierarchy for alt steps, but substeps tend to be rather dissimilar except for the coincidence of being contained in the same plan. Finally, there is a third net for predicates that are used in preconditions, called the P-Net (Figure 6). These nets are used primarily to calculate the similarity of two entries, by taking the product of the weights along a connecting path.

The specific features that Pathfinder currently uses for the various misconception types are summarized in Figure 7. A dash signifies that the corresponding feature is definitely inappropriate for that misconception type and will never be used. The blank entries signify that although Pathfinder does not use the feature, future research might determine that the feature is appropriate for that misconception type. Some misconception probabilities are just implemented as constants, but most are weighted averages of certain features. For example, substituted preconditions use similarity, obscurity, and complexity with respective weights of 10, 1, and 1, so the probability for a substituted precondition x is

$$\frac{10 * \text{sim}(x) + 1 * \text{obsc}(x) + 1 * \text{comp}(x)}{10 + 1 + 1}$$

These factor weights are very *ad hoc* at the moment, and there is no

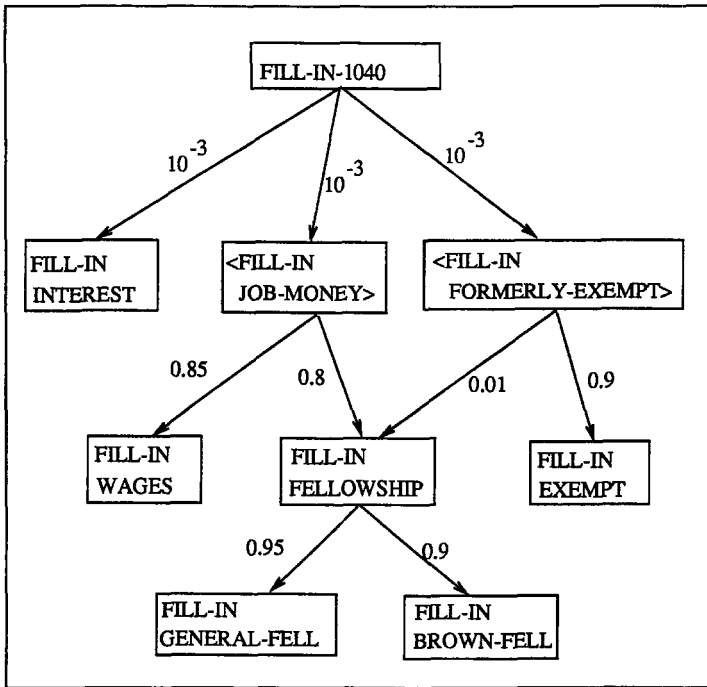


Fig. 5. Sample S-Net.

justification for the current values other than the fact that Pathfinder performed well using them. What is needed here is a stronger cognitive and psychological theory to support the selection and weighting of misconception features.

We can now state exactly what numbers are needed in Pathfinder's user model. The first set of probabilities that we need are the probabilities on the arcs of the plan hierarchy, measuring the probability of selecting a particular alt step given its parent⁵. We also need all the numbers that are required by the misconception scores:

- For each precondition, we need values for similarity, obscurity, and complexity. Similarity is calculated from values in the O-Net, augmented with certain rules to capture syntactic similarity. We also need the similarity value between two substituted preconditions, which is calculated from the P-Net. Complexity is a function of the syntactic form of the precondition, and can be calculated directly. So this only leaves the obscurity value that must be stored for each precondition.
- Violated optimization and violated orderings do not require any values at all, since they get assigned constant scores. Violated bindings are not

⁵ Remember that for the case of substeps, the probability is always a constant 1.

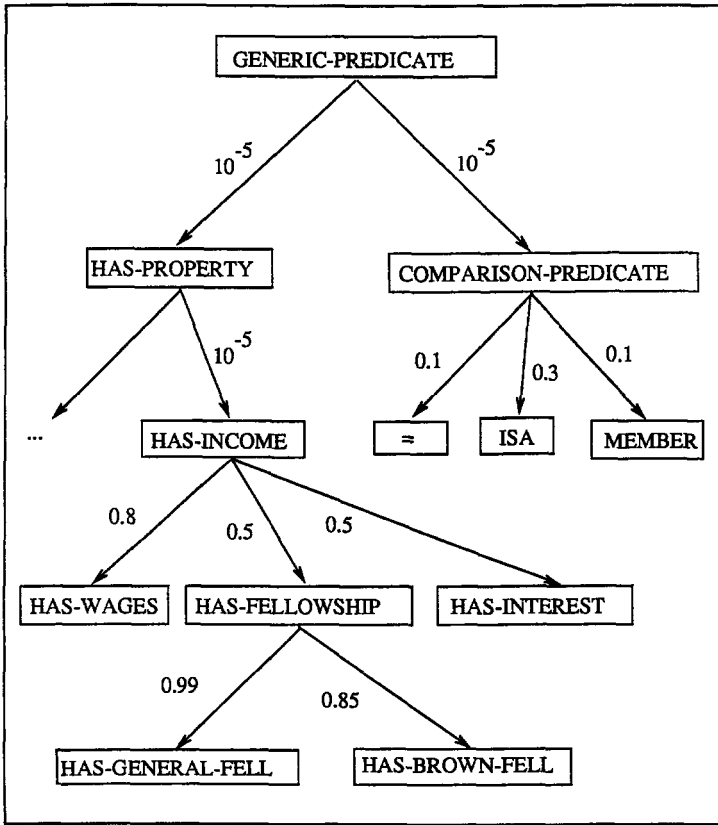


Fig. 6. Sample P-Net.

	VB	VP	SP	MP	EP	OPT	SS	MS	ES	VO
Constant	1	—	—	—	0.001	0.9	—	—	—	0.9
Similarity	—	10	10	—	—		1	—	—	
Obscurity	—	1	1	10			1	1	1	
Complexity	—	1	1	1			1			
Overgeneralization	—								1	
Permanence	—									
Discourse	—									

Fig. 7. Misconception features currently used by Pathfinder.

scored directly; instead, they are translated into violated preconditions of the form (*isa bad-binding good-binding*).

- Substituted and missing steps require scores for obscurity of the correct step, similarity, and complexity. Similarity is calculated from the S-Net, and complexity is a function of the topology of the plan hierarchy. So the only value that needs to be stored is obscurity.
- For extra steps, both obscurity and overgeneralization are functions of the topology of the plan hierarchy, so nothing needs to be stored.

Thus, not counting the information in the three discrimination nets, we only need to store the obscurity value for each precondition and each arc in the hierarchy, and the original correct probability for each alt arc in the hierarchy.

The discrimination nets require quite a bit more information. However, by far the largest of the three nets will be the O-Net, and any intelligent system will already need some sort of *isa* hierarchy as part of its standard knowledge base, so this should not be considered additional information that needs to be maintained. The S-Net will typically have fewer connections than the plan hierarchy itself⁶, and the number of predicates in the P-Net will be negligible compared to the other two nets.

This appears to be a manageable set of numbers. These values could come from a variety of sources. They could reflect the experience of an expert in the domain, who might know from years of consulting that people are quite likely to get W2 forms and 1099 forms confused. Or the values may come from statistics gathered on a large population of users over an extended period of time. For example, the IRS has statistics on how many people file 1040 forms, as opposed to 1040A or 1040EZ forms. Depending on how the statistics are collected, it may be possible to build up profiles of various types of people (for instance, the number of single graduate students with income less than \$20,000 who file a 1040 form). A third possibility would be to use some sort of learning algorithm, where all probabilities for alt steps and misconceptions start off equally weighted, and the probabilities change over time as trends and preferences are observed.

Note that although all of the features are part of the user model, some are more tied to specific individuals than others. For example, **similarity** measures how similar the correct plan component is to the incorrect version that the user has. While there are certainly individual differences in what people consider similar, similarity measures are generally more dependent on the properties of the objects than on properties of the user. So a simple

⁶ The worst case would be if there was no hierarchical structure to the S-Net at all and every pair of steps had a direct similarity connection, resulting in n^2 values for a plan hierarchy of n nodes.

overlay model on top of a default set of values might be appropriate here. On the other hand, **obscurity** measures how familiar the user should be with this particular component of the plan. This is very strongly tied to the individual user; users will have differing backgrounds and experiences that affect their familiarity with plans. So different values will have to be created for each user (or class of users).

7. Analysis

7.1. THE TRANSCRIPTS

One important, but often neglected, aspect of AI research is demonstrating how well the research holds up in the real world. Developing a theory of plan-based misconceptions is not terribly interesting if it does not account for the actual mistakes that people make. And a new algorithm for robust plan recognition is useful only if we can demonstrate that people really do make the sort of mistakes that it is designed to handle.

In order to test both the soundness of our theory of plan-based misconceptions and the practicality of the program described in Section 5.2, we have compiled a large corpus of naturally occurring dialogues, comprising well over 250 pages of text. This corpus includes approximately 100 exchanges that contain examples of plan-based misconceptions (Calistri, 1989). These are examples of actual dialogues that real people have had that contain real plan-based misconceptions. As far as we know, this is the largest set of misconceptions that has been collected to date (an order of magnitude larger than any other known collections).

Two of the goals in creating this corpus were to ensure that the examples came from a wide variety of sources, and to ensure that the examples covered a broad range of topics while still being clustered enough to facilitate an in-depth analysis of a manageable subset.

At the most abstract level, about 90% of the examples come from two general subjects: financial matters and computer systems. These cover approximately 20–30 specific topics. Several of these topics, such as calculating taxes on an IRA, investing in money market certificates, and using the Emacs editor, are well clustered and have many related examples.

The sources of the dialogues are also quite varied. There are five major sources for the examples in this corpus:

- Complete transcripts of the USENET newsgroup “misc.taxes” from the period 7/1/88 to 4/30/89. This is a “bulletin board” style medium, where people post messages over a computer network to ask questions or to

respond to other messages. This particular newsgroup focuses on federal income taxes, but also has some questions about state and local taxes.

- Transcripts of the radio talk show “Harry Gross: Speaking about Your Money”, broadcast on February 1–5, 1982 by radio station WCAU in Philadelphia⁷. This was a radio show where people would call in with questions about how to invest their money, and an expert would advise them. One interesting consequence of this type of medium is that it is in the expert’s best interest to process the maximum number of calls in the shortest possible time, so the expert often tries to diagnose the user’s problem even before the user had completed his first sentence. There are several examples where this led to confusion because the expert applied an incorrect user model.
- Transcripts of a set of on-line dialogues between computer consultants and users of the Emacs editor, collected for a class in spring 1982 at the University of Pennsylvania⁸. These conversations took place using a computerized “talk” program, where the user and the consultant would send messages back and forth to each other. There is one particular set of conversations where several different people have problems with the same concept (marking a region in Emacs), but express their difficulties in significantly different ways.
- Misconception examples that have been used by other researchers. These examples have appeared previously in the literature in various forms; some of them are conversations that actually occurred, some are loosely based around actual situations, and some are completely contrived. These are extracted from the following papers: Joshi et al. (1984a), McCoy (1986, 1989), Pollack (1986), Quilic et al. (1988), and van Beek (1987).
- Miscellaneous examples from the income tax domain that were collected during personal conversations with various people.

Considering the size and diversity of the original text, it should be safe to assume that this represents a good statistical cross-section of the types and frequencies of plan-based misconceptions in real life. These transcripts have served to validate both the developing theory and the Pathfinder program.

7.2. ANALYSIS OF THE CLASSIFICATION

To judge the effectiveness of the classification, we used 97 examples from the transcripts. For each of these dialogues, we located the plan-based misconceptions (some had more than one) and checked to see if they fit into the classification. This is admittedly a rather subjective method, but our goal was

⁷ Thanks to Martha Pollack and Robert Kass for providing these transcripts.

⁸ Thanks to Ethel Schuster for providing these transcripts.

Plan-Based Misconception	Frequency
Violated Binding	16–21
Violated Precondition	7–14
Substituted Precondition	8
Missing Precondition	16–23
Extra Precondition	4–5
Violated Optimization	17
Substituted Step	11–14
Missing Step	5–6
Extra Step	12–14
Violated Ordering	2

Fig. 8. Frequencies of misconceptions in the transcripts.

to try to mimic a “true expert’s” diagnosis and then see if that could be mapped onto our classification. The results of this analysis are shown in the rightmost column of Figure 8. These totals reflect the number of misconceptions (not the number of examples) that fall into each category.

Several of the entries in Figure 8 are ranges instead of single values, because certain misconceptions could fall into more than one category. There are two explanations for this ambiguity. Some of the examples are truly ambiguous, in the sense that there are two possible interpretations that are indistinguishable to an outside observer. Another source of ambiguity is that sometimes the assignment of a misconception to a particular category is dependent on the way that the plan hierarchy is structured. For instance, it is sometimes possible to translate between violated bindings and substituted steps, depending on whether something is encoded as a parameter or as a substep/alt.

There are three examples from the transcripts that do not seem to fit into the classification. In two of the examples, the user thinks there are two marks to be set in Emacs (in fact there is only one), but he never explicitly refers to the second mark so there is nothing we can use to pick out a specific failed component. But the examples are still misconceptions, and they still appear to have a “plan-based” flavor.

The third example that caused difficulty for the classification was a lengthy dialogue involving five different experts. They were discussing how to calculate the capital gains tax on a property sale, and each expert came up with a different answer. Each one pointed out a mistake in the previous expert’s

calculations, then gave her own method for computing the answer. It is possible to find a complex combination of missing and substituted steps, violated bindings, and missing preconditions that explains the mistakes. But a much more natural explanation is that the expert's entire procedure is wrong, and that it should be completely replaced with another method.

Except for these three cases, all other examples from the transcripts seemed to fit within this framework of plan-based misconceptions quite well. The "truly ambiguous" examples, i.e. the examples that people would not be able to distinguish, are correctly treated as ambiguous by the classification. And it turns out that the "structurally ambiguous" examples are actually desirable for the process of automating the detection of these misconceptions, because it often allows a more difficult ill-formed misconception to be handled as an easier violated misconception.

7.3. ANALYSIS OF THE PROGRAM

So far, Pathfinder has been run on 58 different problems (39 examples taken directly from the transcripts, and 19 variations on those examples). In 46 cases (79% of the examples), it returned interpretations that are indistinguishable from what a human observer would conclude. In other words, when the answer was unambiguous, Pathfinder returned the correct answer. And when there were multiple reasonable interpretations, it returned exactly the interpretations that a human observer would, and assigned very believable probabilities to the possible plans.

In 7 additional cases (12% of the examples), Pathfinder also came up with certain technically possible but highly unlikely possibilities that a person might never have considered in the first place. But it still assigned reasonable probabilities to them (on the order of 10^{-5} to 10^{-10}). So these 7 cases should also be judged "successful".

There were 4 examples (7%) where Pathfinder did not assign reasonable probabilities to all of the possibilities. For instance, the example in Figure 9 has an interpretation that the expert has a missing precondition about "safe mode 2". Pathfinder correctly prefers this explanation, with probability 0.76. But it also suggests another explanation: that the expert still has this missing precondition, but is actually trying to *close* the macro instead of opening it, and has a violated binding (using the wrong key) and a violated ordering (trying to close the macro before he opens it). The violated ordering is not penalized strongly enough, so this explanation is considered more likely than it actually should be.

It should be noted that in all 4 of these examples, Pathfinder still correctly prefers the explanation that people find most plausible. It just doesn't assign

- [E] Let's now define a KBD macro. For this one, start at the top of the file. Now, open (or "start remembering") the macro by typing `^X` (....
- [U] `^X` ("is not allowed in safe mode 2".
- [E] Oh, ... my mistake. Most sorry.

Fig. 9. Pathfinder does not assign reasonable probabilities.

- [E] In EMACS you can define what is called a "REGION".... What you do is move the cursor to the beginning or the end of the block, type `esc-M`, and move [to the other end].... OK?
- [U] ... beginning of Region How ?
- [E] Type `esc-M`.... Now end of region. You move the cursor down to the end of the region. OK?
- [U] That's fine, how do I mark, `esc-W` ?
- [E] No, you don't have to mark the end of the region. Only one extreme needs to be marked with the `esc-M`.

Fig. 10. Pathfinder misses the Extra Step.

completely believable numbers to the various possibilities. This is primarily because the methods for calculating the probabilities of misconceptions are only partially implemented.

The only case where Pathfinder came up with the wrong explanation is shown in Figure 10. It correctly catches the violated binding in the user's second question (using `esc-W` instead of `esc-M`). But it misses the extra step of setting the mark for a second time. This is because Pathfinder's temporal reasoning is rather weak. It prefers to think that there are two completely separate `define-region` events happening, which require two `set-mark` steps, rather than the normal interpretation that there is only one `define-region` event which has an extra step. Stronger temporal reasoning abilities will correct this problem.

So in all but one of the cases (98% of the examples), Pathfinder correctly selects the best explanation for the user's plan. It should be noted that Pathfinder achieved this performance using only partially implemented heuristic information; once the remaining heuristic features are added to the program, the probabilities should improve even more. This is an extremely strong argument in favor of probabilistic plan recognition. It is also clear that this is not just a "blocks-world" program; Pathfinder achieved this accuracy while running on 58 real-world problems from several different domains.

7.4. RUNTIME ANALYSIS

As expected, the efficiency of Pathfinder is highly dependent on the accuracy of the heuristics. Like all heuristic searches, it is exponential in the worst case, but can perform very efficiently when provided with accurate measurements. In the worst case, Pathfinder will have to verify

$$1 + 2d \left(f \frac{2d}{m} \right)$$

nodes, where d is the depth of the hierarchy, f is the forward branching factor of the hierarchy, and m is the number of mentioned steps⁹. Since Pathfinder uses mentioned steps to partition the search space, its performance improves as the number of mentioned steps increases. The best case analysis of Pathfinder gives a complexity of $1 + d$ verified nodes, which is achieved when the heuristics give perfect information and when no substituted steps can be made.

Benchmark times were collected from the transcript examples described above. For these benchmark results, Pathfinder was run on a Sun SparcStation with 16 megabytes of memory, using compiled (but non-optimized) lisp code. It used a non-trivial plan hierarchy consisting of all the plans necessary for the 58 running examples. The graph for this hierarchy contains 157 unique nodes. The depth of this graph is 11 nodes, and it has a maximum forward branching factor of 10 and a maximum backward branching factor of 19.

All 58 examples were run completely, generating all possible interpretations rather than stopping as soon as the most likely explanation was reached. 80% of the examples ran in less than 1.0 second, and every example except one ran in under 2.0 seconds¹⁰. Many of these examples were multiply ambiguous; one generated 12 different interpretations (6 of which were given probabilities lower than 0.01) and still ran in only 1 second. When Pathfinder was rerun, stopping as soon as it generated the most likely path, most of the examples ran at least two to three times faster.

8. Conclusions

There is no getting around the fact that people make mistakes, and the cases where people have plan-based misconceptions are exactly the cases where it is critical for an intelligent interface to recognize what is happening. Unfortunately, these cases are also the ones where current methods of plan recognition

⁹ This upper bound will never be reached in practice; it involves a degenerate hierarchy, requires that the heuristics provide no information at all, and requires that all substituted steps are equally likely.

¹⁰ The one example that took longer than 2 seconds was the one shown in Figure 10, which took 4.3 seconds.

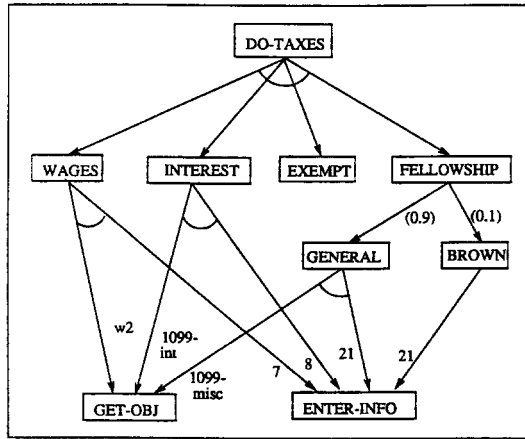


Fig. 11. A simplified hierarchy.

break down most rapidly. There are ways to extend traditional plan recognition techniques to handle this new class of novel flawed plans, but user models are still needed to overcome the high degree of ambiguity inherent in flawed plans.

User models are critical to this process of robust plan recognition, both in providing a means of measuring the relative likelihood of alternative plans, and in calculating the believability of misconceptions that the user may have. Given the classification of a plan-based misconception (which is provided for free during the task of detecting the misconception in the first place), the user model provides a way to measure features like similarity and obscurity of plan components, which gives us the ability to calculate the likelihood of novel misconceptions and of novel flawed plans. By providing these capabilities, user models greatly increase the ability of computer systems to communicate intelligently with humans.

Appendix: A Demonstration

This section simulates the Pathfinder program running on the following query:

“I’m trying to fill out my taxes.
Where do I get the W2 form for my fellowship?”

It uses the hierarchy in Figure 11, which is adapted from Figure 1. The only change is to shorten the hierarchy to focus on the interesting steps relevant to the query.

Pathfinder is given a list of mentioned steps from the parser:

(do-taxes) → ... → (fill-in-fellowship f235)
 → ... → (get-obj form-w2).

It starts out with the top-level goal of *do-taxes* and finds all ways to extend this to *fill-in-fellowship*. In this particular hierarchy, there is only one path that connects the two mentioned steps:

|| *do-taxes* → *fellowship*.

The parallel lines show how much of the path has been verified so far (currently none).

At the same time, it also finds possible substituted steps for *fill-in-fellowship* that would lie along a legitimate path. From the S-Net, we find four reasonable substitutions for filling in fellowship information: *fill-in-general-fellowship* (with a similarity score of 0.95), *fill-in-brown-fellowship* (with a score of 0.9), *fill-in-wages*, (with a score of 0.68), and *fill-in-exempt-income* (with a score of 0.009). But since there is no path connecting *fill-in-exempt-income* with *get-obj*, this path is discarded¹¹. Furthermore, if we replace *fill-in-fellowship* with either *fill-in-general-fellowship* or *fill-in-brown-fellowship*, we will end up duplicating a step that already occurs later in the same path, so these substitutions are also rejected.

However, substituting *fill-in-wages* and *fill-in-fellowship* is acceptable because there is a path from *do-taxes* through *fill-in-wages* to *get-obj*. Assuming that the obscurity and complexity scores for this substitution are 0.05 and 0.1 respectively, this gives us a misconception probability of 0.28 (taking the weighted average of 0.68, 0.05, and 0.1 with equal weights, based on the formula in Figure 7).

Pathfinder makes all calculations on *unnormalized* probabilities, and only delivers a “true” normalized probability at the end of the calculations. While the details are rather complicated, the net effect is that we can multiply all the “raw” probabilities together and do a single normalization at the end, once we get all of the possible steps.

So we now have two partial paths:

|| *do-taxes* → *fellowship* (1.0)
 || *do-taxes* → *wages/fell* → *get-obj* (0.28).

The path for filling in fellowship information currently has an (unnormalized) probability of 1.0, since it contains no alt steps and no misconceptions. The

¹¹ The rationale is that such a path would need at least two substituted steps in a row, or a substituted step immediately followed by an extra step, which is very unlikely. So Pathfinder does not propose such paths in the first place, to save time.

path through `fill-in-wages` has a lower probability of 0.28 (because we proposed a substituted step when we had no particular evidence for it). So Pathfinder will choose the first path and will verify all of the steps along it. There are no violated misconceptions in any of the steps, so the score does not change and this path is still preferred to the path with `fill-in-wages`.

Now Pathfinder expands `fill-in-fellowship` down to `get-obj`. The only way to do this is by passing through `fill-in-general-fellowship`. Since this is a very likely alternative (0.9), and since we still have not encountered any misconceptions along this path, Pathfinder will still prefer it to the explanation with a substituted step. So we have:

`do-taxes` → `fellowship` || → `general` → `get-obj` (0.9)
 || `do-taxes` → `wages/fell` → `get-obj` (0.28).

Pathfinder now verifies the rest of this path. At this point it encounters a misconception: since the student's fellowship is from Brown University and is not a general fellowship, there is a precondition in the `fill-in-general-fellowship` plan that does not hold. The most likely explanation here is that the student was not aware that Brown fellowships didn't apply here, so this shows up as a missing precondition with probability 0.8 (based primarily on the obscurity of the precondition). There is also a violated binding on the form that he is supposed to get: the student mentioned a W2 form, but the correct binding under `fill-in-general-fellowship` is a 1099 form. This gets a probability of 0.72 from the O-Net¹².

This lowers the likelihood of this path to 0.52, and since it involves a misconception, Pathfinder tries to find another possible path connecting `fill-in-fellowship` and `get-obj` that avoids this misconception. This produces a new possibility: the user might be trying to go through `fill-in-brown-fellowship`, which would avoid the misconceptions, but which is normally a less likely alternative (probability 0.1) and would require an extra step (probability 0.5, based primarily on overgeneralization). This now gives:

`do-taxes` → `fellowship` → `general*` → `get-obj*` || (0.52)
 || `do-taxes` → `wages/fell` → `get-obj` (0.28)
`do-taxes` → `fellowship` || → `brown` → `get-obj+` (0.05).

The violated binding and missing precondition are marked with a “*”, and the extra step is marked with a “+”.

¹² As Figure 7 shows, violated bindings actually get a score of 1. The penalty for the misconception comes from interpreting the binding as a set of *isa* constraints and weighting them like violated preconditions.

Since we have now finished verifying one complete path, and since Pathfinder's scores are monotonically decreasing, we have now found the most likely explanation for this particular user's faulty plan. If we continue processing the other two paths, we will not discover any more misconceptions, and the final normalized probabilities will be 0.61 for the missing precondition and violated binding explanation, 0.33 for the extra step explanation, and 0.06 for the substituted step explanation. Since the top explanation is not overwhelmingly preferred, the response generator will probably want to phrase the response in such a way as to cover the second possibility as well.

Acknowledgements

Much of this work was done while the author was at Brown University. Special thanks go to Eugene Charniak. Helpful comments were received from Alfred Kobsa, David Chin, and anonymous reviewers at UMUAI.

References

- Anderson, John R., C. Franklin Boyle, and Gregg Yost: 1985, 'The Geometry Tutor'. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, California, August, pp. 1-7.
- Douglas E. and Martha E. Pollack: 1991, 'Weighted Abduction for Plan Ascription'. *User Modeling and User-Adapted Interaction* 1, in press.
- Brown, John Seely and Richard R. Burton: 1978, 'Diagnostic Models for Procedural Bugs in Basic Mathematical Skills'. *Cognitive Science* 2, 155-192.
- Calistri, Randall J.: 1989, *An Annotated Compendium of Naturally Occurring Plan-Based Misconceptions*. Technical Report CS-89-37, Department of Computer Science, Brown University, Providence, Rhode Island, October 1989.
- Calistri, Randall J.: 1990, *Classifying and Detecting Plan-Based Misconceptions for Robust Plan Recognition*. PhD thesis, Department of Computer Science, Brown University, Providence, Rhode Island, May 1990.
- Calistri-Yeh, Randall J.: 1991, 'An A* Approach to Robust Plan Recognition for Intelligent Interfaces'. In: Nikolaos G. Bourbakis (ed.), *Applications of Learning and Planning Methods*, World Publishing Company, (forthcoming).
- Carroll, Glenn: 1988, Personal correspondence.
- Carberry, Sandra: 1990, 'Incorporating Default Inferences into Plan Recognition'. In: *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, Massachusetts, pp. 471-478.
- Cheikes, Brant A.: 1989, *The Architecture of a Cooperative Respondent*. Technical Report MS-CIS-89-13, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania.
- Goldman, Robert and Eugene Charniak: 1988, 'A Probabilistic ATMS for Plan Recognition'. In: *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, Minnesota.
- Goldman, Robert and Eugene Charniak: 1989, 'A Semantics for Probabilistic Quantifier-Free First-Order Languages, with Particular Application to Story Understanding'. In: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, pp. 1074-1079.

- Goodman, Bradley A. and Diane J. Litman: 1990, 'Plan Recognition for Intelligent Interfaces'. In: *Sixth IEEE Conference on Artificial Intelligence Applications*.
- Goodman, Bradley A. and Diane J. Litman: 1991, 'On the Interaction Between Plan Recognition and Intelligent Interfaces'. *User Modeling and User-Adapted Interaction* 1, in press.
- Hart, P. E., N. J. Nilsson, and B. Raphael: 1968, 'A Formal Basis for the Heuristic Determination of Minimum Cost Paths'. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2), 100-107.
- Joshi, Aravind, Bonnie Webber, and Ralph M. Weischedel: 1984a, 'Default Reasoning in Interaction (Extended Abstract)'. In: *Workshop on Non-Monotonic Reasoning*, pp. 145-150.
- Joshi, Aravind, Bonnie Webber, and Ralph M. Weischedel: 1984b, 'Living Up to Expectations: Computing Expert Responses'. In: *Proceedings of the National Conference on Artificial Intelligence*, Austin, Texas, pp. 169-175.
- Robert and Tim Finin: 1987, 'Rules for the Implicit Acquisition of Knowledge about the User'. In: *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, Washington, pp. 295-300.
- Kass, Robert and Tim Finin: 1988, 'Modelling the User in Natural Language Systems'. *Computational Linguistics* 14(3), 5-22.
- McCoy, Kathleen F.: 1986, 'The ROMPER System: Responding to Object-Related Misconceptions Using Perspective'. In: *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, New York, New York, pp. 97-105.
- McCoy, Kathleen F.: 1989, 'Generating Context Sensitive Responses to Object-Related Misconceptions'. *Artificial Intelligence*, (forthcoming).
- McLoughlin, Henry B.: 1987, 'Personae: Models of Stereotypical Behavior'. In: R.G. Reilly (ed.), *Communication Failure in Dialogue and Discourse*, North-Holland, Amsterdam, Holland, pp. 233-241.
- Paris, Cecile L.: 1985, 'Description Strategies for Naive and Expert Users'. In: *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, Chicago, Illinois, pp. 238-245.
- Paris, Cecile L.: 1988, 'Tailoring Object Descriptions to a User's Level of Expertise'. *Computational Linguistics* 14(3), 64-78.
- Pearl, Judea: 1984, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Publishing Company, Reading, Massachusetts.
- Pollack, Martha E.: 1986, *Inferring Domain Plans in Question-Answering*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania.
- Quilici, Alex, Michael G. Dyer, and Margot Flowers: 1988, 'Recognizing and Responding to Plan-Oriented Misconceptions'. *Computational Linguistics* 14(3), 38-52.
- Rich, Elaine: 1979, 'User Modeling Via Stereotypes'. *Cognitive Science* 3, 329-354.
- Raskutti, Bhavani and Ingrid Zukerman: 1991, 'Generation and Selection of Likely Interpretations During Plan Recognition'. *User Modeling and User-Adapted Interaction* 1, 237-267 (this issue).
- Schuster, Ethel: 1985, 'Grammars as User Models'. In: *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, California, pp. 20-22.
- Sider, Judith Schaffer and John D. Burger: 1991, 'Discourse Understanding in Expert System Interfaces'. *User Modeling and User-Adapted Interaction*, in press.
- Slagle, James R. and Philip Bursky: 1968, 'Experiments with a Multipurpose, Theorem-Proving Heuristic Program'. *Journal of the ACM* 15(1), 85-99.
- van Beek, Peter: 1987, 'A Model for Generating Better Explanations'. In: *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford, California, pp. 215-220.