Classifying and Detecting Plan-Based Misconceptions

Randall J. Calistri
Brown University Department of Computer Science
Providence, RI 02912, U.S.A.

Technical Report No. CS-89-16
February 1989

Classifying and Detecting Plan-Based Misconceptions

Randall J. Calistri
Brown University
Computer Science Dept.
Providence, RI 02912*

February 28, 1989

Abstract

In order to maintain a truly cooperative dialogue, a natural language system must be able to recognize misconceptions in the user's plans, and yet the diversity of mistakes that can occur makes this task appear overwhelming from the very beginning. However, it is possible to organize plan failures into fifteen different categories of plan-based misconceptions. These categories can be further simplified to five detectable types of misconceptions which only need to be checked in a narrow region of the plan hierarchy. This theory is implemented in a program called Pathfinder, which can detect misconceptions from all five simplified classes.

*Some of the work described in this paper was done while the author was working at the GE R&D Center. Support at Brown University has been provided in part by the National Science Foundation under grants IST 8416034 and IST 8515005 and the Office of Naval Research under grant N00014-88-K-0589. The author can be reached at rjc@cs.brown.edu

Contents

1 Introduction

1.1 Why worry about misconceptions?

People are notoriously adept at making mistakes; sometimes they don't know what they want, sometimes they know what they want but don't know how to do it, and sometimes they *think* they know how to do it but it turns out that their plans are flawed because of certain misconceptions that they have.

Most plan recognition systems assume that the agent will have perfect plans. But natural language systems are specifically designed to be used by humans, who are very likely to make mistakes, and the system should be able to recognize these mistakes and correct them. An important class of mistakes occurs when a person has a flawed plan to achieve some goal and communicates part of that plan by asking a question or giving a command.

The long-term objective is to produce a system that will infer an agent's plans based on the questions that he asks, that will check these plans for consistency, correctness, and appropriateness, and that will correct any misconceptions that are indicated by the plans. Such a system will be of great benefit in improving the quality of man-machine interfaces. Some specific results that depend on recognizing misconceptions are the following:

- Programs such as computerized consultants and database query systems will be better able to handle vague, incorrect, or inappropriate questions.
- Intelligent tutoring systems will have an increased capability for identifying the source of the user's mistakes.
- Generalizing from questions to commands, natural language command interpreters (for example, for an operating system) will be better equipped to deal with the frequent novice problem of not knowing how to phrase a request ("do what I mean, not what I say").

1.2 An example

As an example of the type of situations that a system should be able to handle, consider the following hypothetical dialogue between a graduate student (GS) and a tax consultant (TC):

GS: I'm trying to fill out my 1040 income tax form. Where should I list the exemption for my fellowship?

TC: Fellowships are not exempt any more. You should list your fellowship on line 21, but exempt income should not be listed at all.

GS: All right, where do I get the W2 form for my fellowship?

TC: First of all, you want a 1099 form, not a W2 form. And secondly, Brown University doesn't give 1099 forms this year.

GS: You're kidding! I'm going to call the IRS up right now and complain. What's their number?

TC: Well, the number is 1-800-IAM-POOR, but it's 3:00 in the morning and no one will answer now.

This dialogue illustrates the sort of conversations that humans can handle with ease but that computers find very difficult. The difficulty lies in the fact that a technically correct direct answer to the question either does not exist (in the first two cases) or is misleading and inappropriate (in the last case). Instead of blindly answering the question that was asked, the computer must first try to identify the user's plans that motivated the question, and then locate any misconceptions that are present in those plans. If any misconceptions are found, it may be necessary to correct them in the response.

The problem of recognizing misconceptions in users' plans is an important one, and yet it appears to be overwhelming from the very beginning. People are capable of an infinite variety of mistakes, and their plans can be filled with misconceptions at all locations. Any hope of detecting these misconceptions in a systematic way, let alone of deducing the correct plan that the user has in mind, seems remote.

However, it is possible to reduce this problem to a manageable size. Plan failures can be organized into fifteen different categories of plan-based misconceptions. These categories can be further simplified to five detectable classes of misconceptions, which only need to be dealt with in a very small section of the plan hierarchy.

2 Previous work

2.1 Plan recognition with perfect plans

Plan recognition has had a rich and varied history, but the majority of the work has assumed that the agent has perfect plans. This assumption has dominated the work in plan recognition from its very beginning [SD73, SSG78], and has continued to the present time [Cha85, GC88]. [Kau87] contains a good description of some of the variety found in traditional (perfect) plan recognition.

One approach that is more closely related to plan misconceptions is by Carberry [Car86]. While this approach still assumes that the user has perfect plans, it does deal with the issue of plan disparity, i.e. the situation where the plan that the system infers is not the same as the plan that the user actually has in mind. Rather than assuming that the *user* has made a mistake, it assumes that the user is correct and that the *system* has an incorrect model of the user.

2.2 Bug libraries

The main variation on the assumption that users have perfect plans is to keep a list of possible plan violations in a sort of "bug library", which is consulted if the user's actions cannot be accounted for by any of the system's correct plans. This is the approach taken by Sidner and Israel in [SI81] and by Spohrer, Soloway, and Pope in [SSP85]. This method can be useful for improving the efficiency of a system by storing a small set of the most common mistakes; it is particularly common in CAI systems [BB78, Gen79, Joh85]. The obvious problem with this approach, however, is that it can still only recognize the plans

that it specifically knows about (either correct plans or the "buggy" plans that have been explicitly encoded). People are capable of making an infinite number of mistakes, and there is no way to anticipate all of them ahead of time.

2.3 The SPIRIT system

Pollack's SPIRIT system [Pol86] is the first system to successfully address the problem of identifying plan-based misconceptions by reasoning directly about the plans.

Pollack treats plans as mental phenomena, and views "having a plan" as having a certain configuration of beliefs and intentions. In this view, SPIRIT believes that the user has an invalid plan when there are discrepancies between the beliefs it ascribes to the user as a result of having this plan, and the beliefs that the system itself has. Different types of discrepancies correspond to different forms of invalidity. The user's plan is reconstructed by starting with the goal and queried step and applying a set of plan inference rules to try to connect the two. The result is a collection of beliefs that explains what the user's plan is and why he has it. These beliefs are then checked for validity.

The main drawback to the SPIRIT system is that it can only handle a restricted class of plans called "**simple**" plans. These are plans that are composed entirely of **generating** steps. Intuitively, when one step **generates** another, the second step happens immediately without further intervention from the agent. This requires, among other things, that all steps in a plan must be simultaneous.

For example, an agent can have a **simple** plan for asking for Mary's phone number as follows: he can utter the sentence "What is Mary's phone number?", which generates the step of asking the question "What is Mary's phone number?", which generates the step of requesting that information from the listener. However, the agent can *not* have a **simple** plan that also includes calling Mary, since that would take additional intervention on the part of the agent after he finds out her phone number. For the most part, **simple** plans degenerate into single steps that have been "exploded" into more atomic steps that happen simultaneously, such as decomposing the step of turning on a light into flipping the switch and having the light come on.

2.4 The AQUA system

The AQUA system of Quilici, Dyer, and Flowers [QDF88] also deals directly with the issue of detecting plan-oriented misconceptions. They consider three types of beliefs that the user may have about his plan that may be incorrect:

- beliefs about plan applicability conditions (whether a particular plan should be used to achieve a goal)
- beliefs about enablement conditions (whether a particular state must exist before a plan can achieve a goal)
- beliefs about plan effects (whether a state wil exist as a result of a plan's execution)

If the user has any incorrect beliefs about his plan, the system must construct a set of beliefs that expain the user's mistake. These include:

- a system belief that contradicts the user's mistaken belief
- a system belief that can replace the user's mistaken belief
- a set of potential user beliefs that could have motivated the user's mistake

Each class of mistaken belief has its own set of inference rules (and rule schemas) for finding the necessary system beliefs, and its own set of explanation schemas for suggesting potential motivating user beliefs.

A problem with this approach is the dependence on locating the motivating beliefs underlying the user's misconception. The explanation schemas only provide sets of beliefs that the user may *potentially* have; the schemas are incomplete in that they do not cover all possible explanations, and no mechanism is provided for choosing a single explanation in the event that more than one explanation is applicable. In fact, in Section 3.1 we suggest that there are cases in which it is impossible to identify the correct underlying misconception.

In some sense, our work lies in the middle ground between these two systems; it is able to detect misconceptions in a much wider class of plans than SPIRIT without attempting the costly search for nested misconceptions proposed by AQUA.

2.5 Other approaches

Much of the early work on misconceptions had nothing to do with plans at all. The work by Mays [May80] and Kaplan [Kap82] involved two specific types of misconceptions: **presuppositions** and **presumptions**. They were concerned with the cases in which a proposition is entailed by all direct answers to a question. If that proposition turns out to be false, then a direct answer must be avoided because it will produce a misconception.

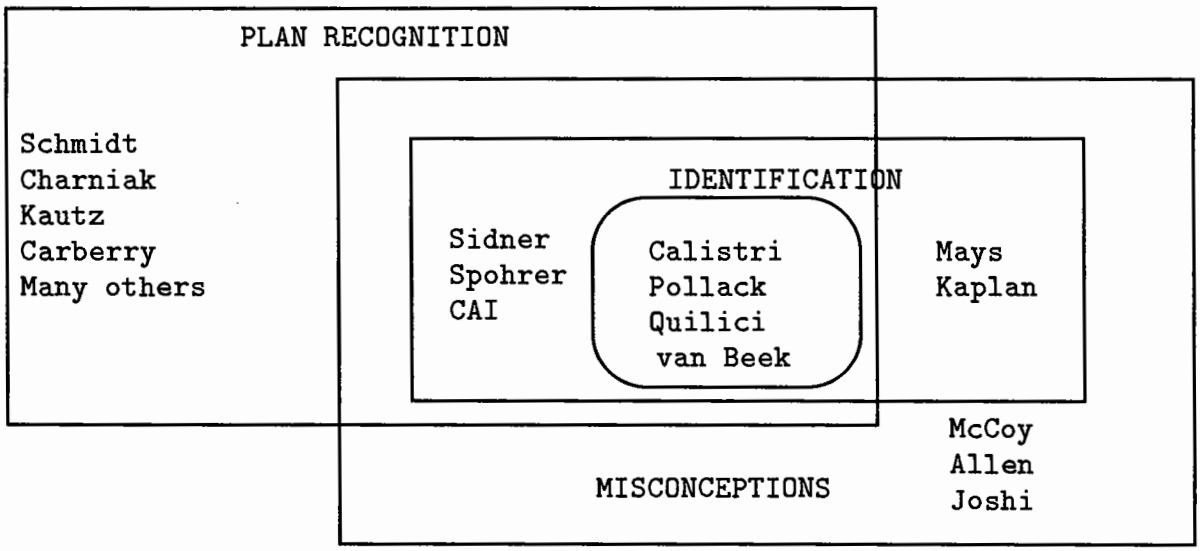
More recently, McCoy [McC86, McC89] addressed the related problem of **misclassification** and **misattribution**. Misclassification occurs in cases where the user has a misconception about how an object should be classified, while misattribution occurs when the user has a misconception about an attribute of an object (either trying to assign an incorrect value to the attribute, or trying to reference a nonexistent attribute). However, her work assumes that the misconception has already been identified, and concentrates on generating an appropriate correction within a specific context.

Additional work on generating responses has been done by [AP80, JWW84b]. They look at misconceptions from the perspective of how to answer a question in the most helpful way. If the user's question contains a misconception, it is necessary to correct that misconception. In addition, if a response is not carefully constructed, it may even cause new misconceptions to occur. Their work in response generation has involved correcting misconceptions that have already been identified and preventing additional misconceptions before they occur, but has again ignored the problem of identifying the misconceptions in the first place.

Other research by [JWW84a] is more directly related to plans. Here they give a classification of instances in which an expert system needs to provide the user with more than a direct answer to his query. This includes cases where the user's plan is either unachievable or is sub-optimal. This work was later extended by van Beek [vB87] to include multiple domain goals and to provide a computational model for verifying the correctness of a plan.

2.6 Summary of previous work

The related work can be summarized in the following chart, which shows the intersection of plan recognition with misconception identification. The circled area contains those systems that do not rely on pre-compiled information about the misconceptions.



3 Plan-based misconceptions

3.1 What is a plan-based misconception?

Before continuing, it is a good idea to clarify exactly what is meant by a plan-based misconception. The term **misconception** refers to an incorrect belief that is held by the user. In contrast, a **plan-based misconception** is a misconception that directly causes a plan to fail. More formally,

> A **plan-based misconception** $\mathcal{M}$ is an incorrect belief held by the user such that
>
> - $\mathcal{M}$ causes plan $\mathcal{P}$ to fail.
> - There is no misconception $\mathcal{M}'$ such that $\mathcal{M}$ causes $\mathcal{M}'$ and $\mathcal{M}'$ causes $\mathcal{P}$ to fail.

The reason for restricting attention to plan-based misconceptions is that "deeper" underlying misconceptions are generally impossible to detect. As an example of why this is so, consider the case of the graduate student requesting the wrong form for his fellowship. Substituting a W2 form for a 1099 form in the `get` step of a plan to fill in fellowship information is a **plan-based misconception**. The reason *why* the user switched them is another level of misconception that can be used to explain the first. There is a potentially infinite number of misconceptions that the user can have at this level. He may think that:

- W2 forms and 1099 forms are the same.
- They are different, but W2 forms are the correct ones to use.
- W2 forms are a special type of 1099 form that are required for fellowships.
- 1099 forms are a special type of W2 form, but any W2 form is acceptable here.
- etc.

Any of these might in turn be explained by a deeper misconception, for example that all forms contain the same information regardless of their number, and so on. Since the depth of this chain of misconceptions has no obvious bound, and since each layer is progressively more concealed from outside observation, the easiest approach is to focus attention on a single conceptual level and deal only with **plan-based misconceptions** in the first place.

Of course, we might know something about the user or about the particular situation that would lead us to prefer one of these beliefs above 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. But it is not possible to rely on having such detailed information all the time. The important thing to note is that all of these potential incorrect beliefs have one thing in common: the user is somehow confused about the relationship between W2 forms and 1099 forms, and this confusion led him to choose the wrong one. This is precisely the plan-based misconception that the user has, and this is the only information that is needed to correct the immediate problem with his plan (in many cases this information will also be sufficient to allow the user to correct his underlying incorrect beliefs himself, whatever they may be).

3.2 What do plans look like?

In order to classify the various types of plan-based misconceptions, it is necessary to define the structure of plans, and to define the hierarchical arrangement of plans.

The plans that we consider here have the following components:

- **parms** are the parameters of the plan. If the plan is viewed as a frame, these are the slots. A **plan schema** is a stored template for a plan, which has variables for most of its **parms**. By convention, variable parms always begin with a "?". When a plan is invoked, many of these variables get **bindings** to specific objects. For example, a plan to `get` an object might have **parms** of `?what`, `?from`, and `?to`, and might have **bindings** of `object84`, `current-loc273`, and `new-loc5` respectively.

- **preconditions** are statements that must be true for the plan to be executed, and determine whether the plan should even be considered by the user. These are assumed to be simple propositions about the state of the world, which can be checked by querying a database of facts. These are similar to **applicability conditions** [Car88] in that the user cannot attempt to satisfy the conditions himself. Typical preconditions are often time-invariant (such as a precondition that the agent of some action must be animate) or beyond the control of the user (such as a precondition that a fellowship cannot be

from Brown University in order to get a 1099 form)[1]. Any condition that the user *can* control will be represented as a **substep**, since the user must have a plan for achieving these conditions.

- **substeps** are necessary and sufficient steps that must be carried out in order to do the plan. These steps may be atomic actions, or they may be complex subplans. In the tax domain, the substeps for filling in earned income may consist of getting a W2 form from your employer, reading the amount on the form, and entering it on line 7 of the 1040 form.

- **alts** are alternate ways of doing a plan. For example, there are two ways to fill out your taxes: either do it yourself, or have a tax service do it for you. If **alts** are present, then **steps** must be `nil` (and vice versa).

- **add** and **delete** are used for atomic actions. If both **steps** and **alts** are `nil`, then the plan is assumed to take place without further decomposition (no subplan is needed), and **add** and **delete** update the state of the world to reflect the effects of the action.

- **orderings** are temporal constraints placed on the order of execution of steps. These are nil for **alts**, and enforce a partial order on **substeps** and **atomic actions**.

A set of plans can be seen to form a hierarchy resembling an **and/or** tree[2]. The leaves of the tree correspond to atomic actions. Plans which have **substeps** form the **and** nodes of the tree ("you need to do A *and* B *and* C"). Plans consisting of **alts** comprise the **or** nodes of the tree ("you can either do it this way *or* that way). In theory, the plan hierarchy is strictly interleaved, alternating between layers of **substep** nodes and layers of **alt** nodes. In practice, however, a layer containing only a single node can be collapsed into the parent node.

Note that there are actually *two* plan hierarchies: the system's hierarchy, which contains all of the correct plans for the domain, and the user's hierarchy, which contains the user's (possibly flawed) versions of these plans. We assume that these two hierarchies have the same general structure; in fact, the user's hierarchy is assumed be identical to the system's hierarchy except for the flaws caused by the user's misconceptions. We are not making any claims as to whether the user *actually represents* his plans in this way; it is sufficient that the system's plan hierarchy encodes the same information as the user's collection of knowledge about how to achieve his goals.

Some examples of plans follow:

```
(def-plan
  :name 'fill-in-1040
  :substeps '((fill-in-earned-income (earned-income-of *you))
```

[1]Note that the second example is technically within the control of the user; he could give up his lucritive Brown fellowship and take a much smaller fellowship from the University of Southern North Dakota at Hoople just to get a 1099 form, but this is obviously something that should not be done.

[2]This is technically a *dag*, not a *tree*, since a step can appear in more than one plan. However, it can be transformed into a tree in the obvious way, by duplicating the step each time it is used in a different plan.

```
              (fill-in-interest (interest-of *you))
              (fill-in-fellowship (fellowship-of *you))
              (fill-in-exempt-income (exempt-income-of *you))))

(def-plan
  :name 'fill-in-fellowship
  :parms '(?source)
  :preconditions '((isa ?source 'fellowship)
                   (has-income *you ?source))
  :alts '((fill-in-general-fellowship ?source)
          (fill-in-brown-fellowship ?source)))

(def-plan
  :name 'fill-in-general-fellowship
  :parms '(?source)
  :preconditions '((not (equal (university-of ?source) 'brown)))
  :substeps '((get 'form-1099 (university-of ?source) *here)
              (read-amt ?amt 'form-1099)
              (enter-info ?amt 'line-21 'form-1040))
  :orderings 'total)
```

3.3 How can plans fail?

A theory of plan-based misconceptions must take into account all ways that any part of a plan can fail. Any of the plan components presented in Section 3.2 can contain misconceptions, and there are several different ways that each of them can fail. For the purposes of classification, it is convenient to group **substeps**, **alts**, and atomic actions together and simply refer to them as **steps**, since only one of them can be present in any given plan anyway. Also, this paper will refer to **binding** misconceptions rather than **parm** misconceptions, associating the misconception with the *value* of the parameter rather than with the parameter itself. This gives four basic components of a plan that can fail: **bindings**, **preconditions**, **steps** (including substeps, alts, and atomic actions), and **orderings**.

Each of these components can fail in any of four ways:

- A **violated** component occurs when the user has the correct component, but it has not been satisfied properly.
- A **substituted** component occurs when a component's correct value has been replaced by an incorrect value.
- A **missing** component occurs when the user should have included this component in his plan but didn't.
- A **extra** component occurs when the user includes this component in his plan but shouldn't have.

This is summarized in the following chart.[3]

PLAN-BASED MISCONCEPTIONS	
Component	**Failure**
Bindings	Violated Binding Substituted Binding Missing Binding Extra Binding
Preconditions	Violated Precondition Substituted Precondition Missing Precondition Extra Precondition
Steps	Substituted Step Missing Step Extra Step
Orderings	Violated Ordering Substituted Ordering Extra Ordering Missing Ordering

Binding failures

There are four different ways that **bindings** can fail:

a. Violated binding. Certain plan parameters can be restricted by the calling plan. If the user attempts to use an incompatible binding, a violation occurs. For example, when the `dial` step is used by plan to call the IRS, the `?number` parameter will automatically be bound to the phone number for the IRS. If the user tries to supply a different number, it will violate the binding that already exists.

b. Substituted binding. Even if the user gets the right values, he may try to bind them to the wrong parameters. For example, in attempting to **get** a certain item, he may reverse the source and destination.

c. Missing binding. A plan may actually have n parameters, but the user may think that it has less than n. For example, in a plan for investing his money, he may ask what the current interest rate is for a CD, not realizing that the rate depends on the length of investment.

d. Extra binding. A plan may have n parameters, but the user may specify more than n. He may think that besides having the phone number, dialing also requires specifying the phase of the moon.

[3] **Violated steps** are purposely omitted,and will be discussed in Section 3.3.

Precondition failures

As with bindings, **precondition** can also fail in four different ways:

a. **Violated precondition.** Since a precondition is something that must be true for a step to be executed, a precondition whose value is false when the user tries to execute the step is an obvious failure. For example, in order for the user to **get** a 1099 form for his fellowship, the fellowship must not be from Brown University. If his fellowship was from Brown, then the step will not succeed.

b. **Substituted precondition.** In this case, the user is aware of the precondition, but his version of the precondition is incorrect. For example, he may think that he gets a 1099 form anyway as long as the fellowship was awarded after 1986.

c. **Missing precondition.** Here, the user 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 user does not even know that the precondition exists ("I didn't know that Brown didn't give out 1099 forms"). With a **violated precondition**, the user 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").

d. **Extra precondition.** Here, the user 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 user might also think that he has to be a first-year graduate student in order to get a 1099 form.

Step failures

We will now try to break **step** failures (including **substeps, alts,** and atomic actions) into the same four classes:

a. **Violated step.** This is the only combination of component and failure that cannot occur. Since a step is itself a plan, any violation of a step is actually a plan failure, and will be recognized as one of the other types of plan failures. Therefore, this type of failure can be ignored.

b. **Substituted step.** The user is attempting an incorrect step in place of the correct step. An example of this would be if the user tried to record the money from his fellowship by using `fill-in-earned-income` instead of `fill-in-fellowship`.

c. **Missing step.** The user should have included this step in his plan, but didn't. There are three versions of this, corresponding to the three types of steps. Missing **substeps** and **atomic actions** (**add** and **delete** steps) are the simplest to understand; the user wasn't aware that it was necessary to do this step in order to complete his plan. For example, he might not know that fellowship information has to be filled in when completing his 1040 form. Missing **alt** steps are a bit trickier. An example of this would

be if the user didn't know that there was a difference between general fellowships and Brown fellowships. He would only have a plan for one or the other, and so would not have an **alt** node to distinguish between them.

d. Extra step. The user is including a step that doesn't belong. These are exactly analogous to **missing steps.**

Ordering failures

Orderings can be viewed as a special type of **precondition**, specifying that one step must precede another. So they can have the same types of failures.

a. Violated ordering. The user thinks (correctly) that step *a* has be done before step *b*, but thinks (incorrectly) that step *a* has already been done.

b. Substituted ordering. The user thinks that step *a* has to be done before step *c*, but it really has to be done before step *b*.

c. Missing ordering. The user doesn't even know that step *a* has to come before step *b*.

d. Extra ordering. The user thinks that step *a* has to come before step *c* too, but it doesn't.

4 Detectable plan-based misconceptions

The fifteen classes of plan failures presented above give a complete classification of misconceptions for the types of plans described in Section 3.2. However, this classification is unnecessarily complex: certain misconceptions are undetectable, and others cannot be distinguished from one another without "looking inside the user's mind". There is no point in keeping undetectable misconceptions in the classification, and classes that are indistinguishable from each other can be merged into a single new class.

To avoid the complications associated with discourse and ongoing conversations, this paper makes the simplifying assumption that the user's queries occur in isolation with no history of previous queries or actions.

4.1 Simplifying the search space

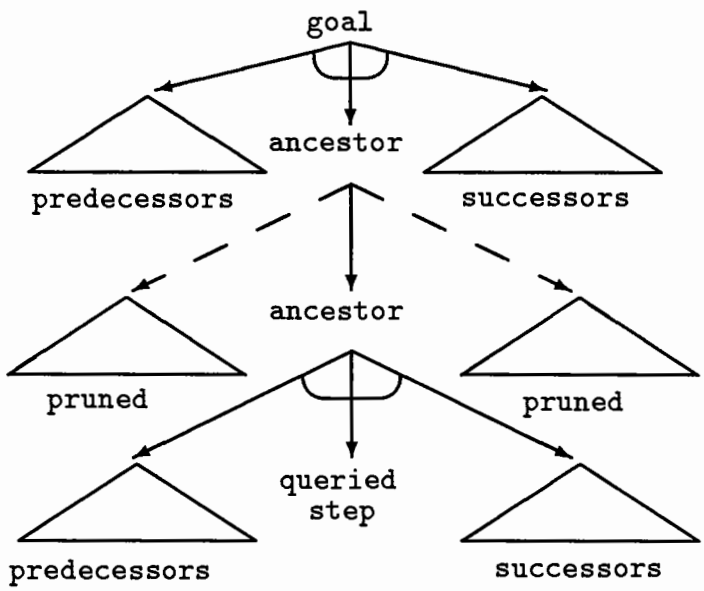
Most types of misconceptions can occur anywhere in the plan hierarchy, although they are undetectable in all but a very small part of the hierarchy. The intuitive reason for this is that the user generally only provides information about the specific steps mentioned in his query. The system will not have any knowledge about misconceptions that occur in parts of the plan that the user is not discussing.

To be more accurate, let us break the plan hierarchy up into several sections and examine each of them to determine whether misconceptions can exist in this area and whether they have any hope of being detected. To illustrate the dimensions of the sections, we will use the following user query:

"I'm trying to fill out my taxes. Where do I get the W2 form for my fellowship?"

The plan hierarchy is first pruned to limit it to plans related to the query. In other words, any **alt** nodes (choice points) that have already had one of their children selected by the query will have all other children (and the subtrees rooted there) removed. In this example, since the user has decided to fill out his taxes, any other alternate plans that he may have had (writing a symphony, taking a bath, etc.) can be eliminated.

The pruned plan hierarchy can then be partitioned into five sections as follows (arcs are used to mark the **substep** nodes).



- The **queried step** is the most specific step that can be extracted from the query, in this case `(get form-W2 ?where *here)`.

  The one type of misconception that cannot occur at a **queried step** is a **missing step**. Since the user specifically mentioned the **queried step**, it is clearly not possible for it to be missing (it may be wrong, but it has to be there). Other than this, all other misconceptions can occur here and have a chance of being detected.

- The **goal** is the most general step that can be extracted from the query, in this case `(do-taxes)`. Alternately, it could be assumed that there is always a constant top-level goal of `(generic-plan)`.

  For the same reason that the **queried step** cannot be missing, the **goal** must also be present. All misconception types other than **missing step** are allowed and have a chance of being detected.

- The **ancestors** are all nodes that lie along the vertical path between the **goal** (inclusive) and the **queried step** (exclusive). Some of these **ancestors** may have been

specifically mentioned by the user (such as `fill-in-fellowship` in this example), but others may be unmentioned.

Again, the one type of misconception that causes problems is **missing steps**. There cannot be a **missing step** misconception at any of the *mentioned* ancestors, for the same reason that the **queried step** can't be missing.

Steps that are *not* mentioned by the user, on the other hand, can be missing, but these misconceptions are undetectable: since the user never mentioned the step in question, there is no way for an outside observer to decide whether he is including that step in his plan, or whether he has "short-circuited" his plan to skip over that step.

With the exception of the **mentioned steps**, all other misconception types can occur at **ancestor** nodes and have a chance of being detected.

- The **predecessors** are all nodes that temporally precede the **queried step**. This basically includes all nodes to the left of the vertical path (assuming that steps are listed in chronological order from left to right), such as getting the 1040 form (note that the vertical path itself represents a single time period). Steps that have no temporal order relative to the **queried step** can be included in either **predecessors** or **successors**; as we shall see, it will not make any difference.

  Any type of misconception (including **missing steps**) can occur at a predecessor node. However, in the case of isolated queries, they are almost all undetectable. The intuitive reason for this is that the system does not have any knowledge about any steps other than the **queried step** and the **ancestors**. To take **extra step** misconceptions as an example, the user might think that before he fills in his fellowship information, he has to order a pizza. But unless he mentions that fact, the system will never be made aware of it. All **binding** misconceptions and **step** misconceptions work in this way.

  In fact, in general the system does not even know whether a given node is part of the user's current plan or not. Remember that a substantial percentage of the plan hierarchy consists of **alt** nodes, which function as choice points. The user has presumably already chosen which of the alternate subplans he intends to use for many of these choice points, but since this is not part of the query, the system has no way of knowing how the plan hierarchy has been pruned. This prevents the system from checking **preconditions** (a misconception type that could normally be detected) because even if it finds one that has been violated, it doesn't know whether or not the user is going to try to execute that particular step (it might be part of the hierarchy that the user already pruned off).

  There is one partial exception to this: there is one case in which the system *does* have strong reason to believe that the user intends to do certain steps. These steps are precisely the siblings of any **substeps** in the vertical path from the goal to the queried step, which are guaranteed to be part of the user's current plan (another way of describing these same steps is as the children of any **alts** in the vertical path, assuming that **substep** and **alt** layers are strictly interleaved). However, this is not true for the descendants of these steps; as soon as an **alt** step is encountered, we are back in the problem of the last paragraph.

It is unclear how **predecessor** misconception detection will be affected by discourse and ongoing conversations. This will certainly provide more information about the user's past actions, but there will still be many situations in which not enough information is available.

- The **successors** are all nodes that follow the **queried step** in time, and covers the right hand side of the plan hierarchy.

  Misconceptions located at **successor** nodes are exactly analogous to the case with **predecessors**; in general, there is no way to tell whether a given **successor** node is part of the user's current plan or not.

Therefore, with the isolated-query assumption, the only section of the plan hierarchy that has any hope of containing consistently detectable plan-based misconceptions is the single vertical path from the **goal** to the **queried step**.

Of course, there are instances in which specific knowledge about the user's previous actions or knowledge about the ongoing discourse would provide evidence for a misconception in a **predecessor** or **successor** step. It is likely that methods for recognizing certain of these clues can be automated; additional research is needed in this area to determine how prevalent these cases are and how systematically they can be handled.

4.2 Simplifying the classification

The beginning of Section 4 claimed that certain misconception categories are undetectable, and that others, although distinct within the mind of the user, are indistinguishable by an outside system. This is caused by the fact that the system does not have the opportunity to observe the user's misconceptions directly; it can only observe the effects that these misconceptions have on the user's query.

Each type of misconception will now be examined to determine which categories are detectable and distinguishable. Undetectable categories can be ignored, and indistinguishable categories can be collapsed together.

Bindings

Bindings can be simplified into two categories.

Missing and **extra bindings** both occur when the user has the wrong number of parameters for a particular plan. The problem of logic representation caused by having a wrong number of parameters will be automatically taken care of in the process of translating from the user's English input to the internal representation of plan invocations. This translation mechanism knows about the number of parameters that each plan requires, and it will attempt to find values to bind to each of these parameters in the user's query (or, failing that, will attempt to find a default value for them). Parameters that still have no value (i.e. parameters that the user is missing) will be filled in with Skolem constants [Cha88]. In this way, the plan is guaranteed to have the correct number of parameters.

The question remains whether or not **missing** or **extra bindings** will actually occur in real life. As far as **extra bindings** are concerned, the translation mechanism would

have no reason to attach any extra parameters to the plan. Although it is easy to come up with realistic English queries that contain extraneous information, it is surprisingly difficult to come up with an example where that extra information unambiguously entails an extra parameter to the plan. **Extra bindings** will therefore be considered undetectable.

Missing bindings are a little trickier. There are borderline examples such as investing in a CD from Section 3.3. But it is unclear whether this is truly a **missing binding** or whether the user is assuming a default value for the binding. If **missing bindings** do exist, they are identifiable by the fact that an unbound Skolem constant still remains in an instantiated plan (either because the user did not provide the necessary value, or because no default value could be found). We will leave **missing bindings** as a separate class for now, but it may be eliminated in later versions of this paper.

Although **Substituted bindings** *can* be treated easily enough as multiple occurrences of **violated bindings**, it would be nice to handle them separately since two bindings that are simply in the wrong order should not be penalized as much as two bindings that are each incorrect on their own. It would appear at first glance that **substituted bindings** can in fact be handled separately: since two bindings have been switched, try to "unswitch" them by permuting pairs of bindings until the violations are eliminated.

However, it is generally impossible to sort out which bindings have been substituted. It is not sufficient to just try switching pairs of incorrect bindings until the violations are eliminated. First of all, the substitutions may not be strictly pairwise — there may have been three or more bindings involved in the substitution. Secondly, the substitution may involve one violated binding and one *correct* binding (as in the case where one of the plan parameters had no restrictions placed on it, so no violation occurred). Finally, one or both of the user's bindings may have been incorrect even before the substitution, so even after the substitution is undone violations may still occur.

Therefore, **substituted bindings** are indistinguishable from **violated bindings**, and the two categories can be collapsed together. This leaves **violated** and **missing bindings** as the only two types of detectable binding misconceptions.

Preconditions

Of the **precondition** misconceptions, only **violated preconditions** need to be considered.

Extra preconditions are simply undetectable; the user is not explicitly mentioning any preconditions, so there is no way of knowing whether he has any extra ones.

Missing and **substituted preconditions** can be collapsed into **violated preconditions**. All will be observed as violations of the correct precondition whether the user had the correct precondition, whether he was missing the precondition completely, or whether he had a different precondition in its place. This is again because the user does not explicitly mention the precondition; the only thing that an outside observer can see is that the correct precondition has been violated. He can not see what precondition, if any, is included in the user's version of the plan. Note, however, that **missing** or **substituted** preconditions that are satisfied by coincidence will go undetected.

This leaves us with **violated preconditions** as the only detectable type of precondition misconceptions.

Steps

Violated steps have already been eliminated from the classification. An analysis of Section 4.1 will reveal that **missing steps** have been proven impossible or undetectable in all sections of the plan hierarchy.

Substituted steps can sometimes, but not always, be distinguished from **extra steps**. The deciding factor is whether or not the correct step that the **substituted step** has replaced can be identified. If an observer cannot find the corresponding step, then he has no choice except to assume that it is an **extra step**, even though it may have actually been a **substituted step** from the user's perspective.

There are certain situations where it is easy to find the corresponding step (for example, when a mentioned substep is present in both the plan for the **substituted step** and the corresponding step). Similarly, there are other cases in which the correct choice is an **extra step** (for example, when the step in question is an atomic action that is unrelated to any candidate substitutions). However, there is a poorly-defined gray area where the choice between **substituted step** and **extra step** is not so clear-cut; some people would be able to figure out the user's substitution, and others would not. More research needs to be done to better define the boundaries of this ambiguous area and to decide what to do for the cases that fall inside it.

Regardless of this, however, **substituted** and **extra steps** are the only two categories that are both detectable and (at least sometimes) distinguishable.

Orderings

Section 3.3 stated that **orderings** could be viewed as special types of **preconditions**. And, just like **preconditions**, **extra orderings** are undetectable and **violated, substituted**, and **missing orderings** can be collapsed into a single category.

However, it is not clear whether this simplified category is detectable. Although **orderings** are basically specialized **preconditions**, there is one crucial difference. Since **preconditions** are assumed to be simple propositions about the state of the world, it is easy to verify whether a precondition has been satisfied. There is no corresponding way to verify whether an **ordering** has been satisfied.

Recall that this paper assumes that the user's queries occur in isolation with no knowledge of previous actions or discourse. This means that we know the current state of the world, but not how that state was reached (i.e. not the sequence of steps that caused this state). Without explicit knowledge of the times at which specific steps were executed, it is often impossible to tell whether a given step has occurred before another step, or even whether it has taken place at all. This is because some of the effects of a step may be cancelled out by later steps.

To illustrate, let us say that step A has effects x and $\neg y$, that step B has effects y and z, and that step C has effect y. If x, y, and z are all satisfied in the current state of the world, there is no way to choose between the sequence of events $[A \rightarrow B]$ and $[B \rightarrow A \rightarrow C]$. In other words, there is no way to tell whether A preceded B or whether B preceded A. Furthermore, if there is another step D that can also cause the effect x, it is possible that step A was never executed at all.

There are obviously many situations in which an unambiguous order can be assigned to two steps that have occurred. However, further research is needed in this area to characterize the circumstances under which **ordering** misconceptions can be accurately detected, and to determine what to do in the case of ambiguity. In many applications, incorrect detection of a misconception that does not exist can be extremely confusing to the user, and may end up being more detrimental than missing a misconception that actually does exist.

Summary of classification

The following chart shows the simplifications that have just been made:

PLAN-BASED MISCONCEPTIONS		
Component	**Failure**	**Simplification**
Bindings	Violated Substituted	Violated Binding
	Missing	Missing Binding
	Extra	XXXXX
Preconditions	Violated Substituted Missing	Violated Precondition
	Extra	XXXXX
Steps	Substituted	Substituted Step
	Missing	XXXXX
	Extra	Extra Step
Orderings	Violated Substituted Missing	?????
	Extra	XXXXX

5 The Pathfinder program

5.1 What is Pathfinder?

The theory of the previous sections has been implemented in a program called Pathfinder. Pathfinder is designed to be a component of an integrated natural language system. Its purpose is to act as an interface between the semantic interpreter and an application program such as a command executor or database query program. Rather than just blindly executing the user's commands or retrieving the information that the user requested, Pathfinder will analyze the user's goals and his command or query and attempt to determine if the request the user is making is reasonable.

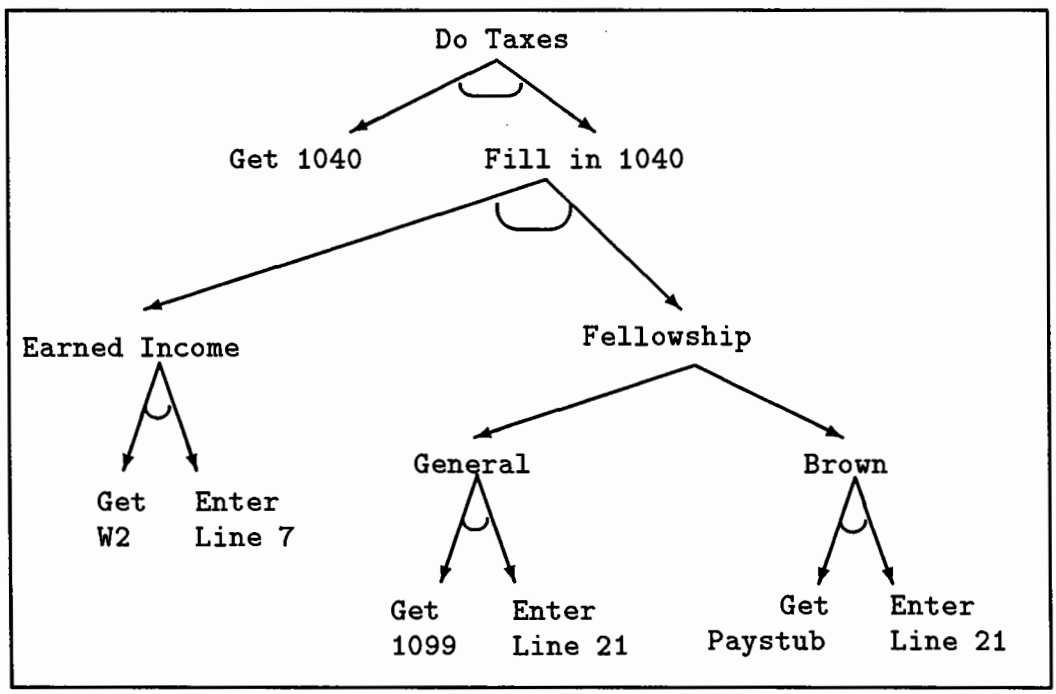
Input to the Pathfinder program is an incomplete partial path leading from the user's top-level goal to the specific step that the user is asking about. The job of Pathfinder is to expand this path, filling in the intermediate goals and steps, and to verify that it is part of

a correct plan. Its output is this expanded path, marked with appropriate error messages for any misconceptions that have been detected.

5.2 Expanding the partial path

Pathfinder has access to a hierarchy containing all correct plans of the domain. It tries to locate the partial path in this plan hierarchy by constructing a complete path between each pair of consecutive steps in the partial path.

The following is a simplified portion of a plan hierarchy from the tax domain[4].



A query of "I'm trying to fill out my taxes. Where do I get the W2 form for my fellowship?" might be passed to Pathfinder as the partial path

$$\texttt{(do-taxes)} \rightarrow \cdots \rightarrow \texttt{(fill-in-fellowship)} \rightarrow \cdots \rightarrow \texttt{(get form-w2)}$$

It will search the plan hierarchy bottom-up, starting with the queried step of `(get form-w2)`, and attempt to locate a path up to `(fill-in-fellowship)`. Then it will try to continue to expand this path from `(fill-in-fellowship)` up to `(do-taxes)`. In this case, the first part of the expansion cannot be done correctly, signalling that the user has a misconception. The second part of the expansion can be done by passing through `(fill-in-1040)`.

[4]Nodes with **substeps** are marked with arcs; nodes with **alts** are not. Nodes that are not needed for the following examples are not shown here in order to save space.

5.3 Finding violated preconditions and bindings

As the expanded path between each pair of steps is being constructed, it is checked for misconceptions. First, the **preconditions** of each step are verified.

In addition to checking the preconditions, parameters are also checked to make sure that they are bound correctly. Note that significant **violated bindings** only occur at **mentioned steps**; any other **violated bindings** will just be "cascading" consequences of an earlier violation. Therefore, it is only necessary to check the endpoint of the path. The bindings are checked by comparing the parameters specified by the user with any parameters that are preset in the parent's stored plan. Any pairs that cannot be unified are considered to be **violated bindings**. Additionally, any parameters that should have been bound by this point but haven't been are treated as **missing bindings**.

5.4 Finding incorrect steps

It may be the case that an expanded path between two mentioned steps cannot be found. This means that the user claims there is a path between two steps, but the system's plan hierarchy does not contain any such path. The user must therefore have an error in the connections of his plan hierarchy, signifying either a **substituted step** or an **extra step**.

The difference between these two is that with a **substituted step**, the user has replaced a correct step with an incorrect step, and is now off on the wrong path. In contrast, with an **extra step**, the user has simply inserted an additional step into an otherwise correct path.

It is easy enough to figure out that the user has an incorrect step; this situation is signalled by the fact that no path can be found to connect the two mentioned steps. Deciding whether the mentioned step should be replaced with something else or completely ignored, however, seems to be a more challenging problem. Putting this problem aside for the moment and assuming that the type of incorrect step has already been identified, it is fairly simple to handle each case individually.

As an example, assume that the user's input is translated to a partial path of

$$a \rightarrow \cdots \rightarrow b \rightarrow \cdots \rightarrow c \rightarrow \cdots \rightarrow d \rightarrow \cdots \rightarrow e$$

(where a is the top-level goal and e is the queried step), and assume that Pathfinder has already expanded the path from e up to d. It will then continue by trying to expand the path from d up to c. If this fails, it is either because of a **substituted step** or an **extra step**.

Assume that d has been identified as an **extra step**. Then an appropriate error message will be inserted, d (and its entire subplan, including e) will subsequently be ignored, and Pathfinder will continue on trying to expand the path from c up to b.

Alternately, assume that c has been identified as a **substituted step**. Then rather than trying to connect d to c, Pathfinder will ignore d, and will instead try to find a different path to c. It will then insert an error message stating that instead of doing step d, the user should have done the corresponding step in the new path. The selection of the appropriate new path is obviously important. Currently, Pathfinder tries to expand all possible paths from e to c (ignoring d), on the assumption that the correct path will still contain e. This will not always work, but it is a reasonable first approximation.

5.5 Multiple possible paths

Substituted steps are not only proposed when it is impossible to find an expanded path; they should also proposed when a path is found that already contains misconceptions.

As an example, suppose the user said

> "I'm trying to fill out my taxes. Where do I get the W2 form for my fellowship?"

It is possible to reconstruct a complete path through

`(do-taxes)` $\rightarrow \cdots \rightarrow$ `(fill-in-fellowship fell235)` $\rightarrow \cdots \rightarrow$ `(get form-w2)`

The last step of this path contains a **violated binding** (using `form-w2` instead of `form-1099`). However, it is also possible to construct a *different* path:

`(do-taxes)` $\rightarrow \cdots \rightarrow$ `(fill-in-earned-income fell235)` $\rightarrow \cdots \rightarrow$ `(get form-w2)`

This path avoids the earlier **violated binding**, but now contains a **substituted step** (substituting `fill-in-earned-income` for `fill-in-fellowship`) and a **violated precondition** (`fell235` is not earned income).

These two paths suggest two different scenarios:

- The user got W2 forms and 1099 forms confused, and he was expecting the wrong one.
- The user knows the difference between W2 forms and 1099 forms, but he thinks that fellowships are a form of earned income (which *does* get a W2 form).

In order to capture these two possibilities, Pathfinder looks for possible **substituted steps** whenever a misconception is found in the construction of the expanded path. The motivation behind this is that once a misconception is found, there is reason to suspect the current path. The user has obviously made a mistake by this point; it may be that we were lucky enough to recognize it at the exact spot that it was made, or it may be that the user already wandered off the correct path earlier and this error is a signal that he's on the wrong path.

This creates a problem, since there are now multiple possible paths for a single user input. This means that there is more than one possible explanation for the user's mistake. However, only one of them is the *real* explanation, and it is very important to choose the correct one.

To illustrate the potential danger, reconsider the previous example. A correct answer for the first scenario would simply point out the fact that the user should get a 1099 form, not a W2 form. But this answer would just add to the user's confusion in the second scenario; instead of correcting the problem, it might end up giving the user another misconception, and now he would think that *all* earned income is supposed to get 1099 forms.

It should be possible to solve this problem by associating a score with each misconception that is found. This score would measure the likelihood that the user has this particular misconception. This provides a way to select a single path from all the alternates; the path with the highest total score is the most likely. This also provides a method for reducing the search problem, by using the scores to guide a best-first search. Since the final answer will be the path with the best score, it is only necessary to extend the path that is currently judged

to be the most likely. Even though many potential alternate paths may exist, these new branches will only be explored if the original path becomes relatively unlikely in comparison.

However, this raises a new concern. Best-first search must be guided by a monotonic heuristic function, and it is not obvious that these scores will provide such a function. One possibility is to interpret the scores as probabilities: a path initially starts out with a score of 1, and this is multiplied by the score of each misconception that is found. Since all probabilities must be between 0 and 1, this appears to be monotonically decreasing, as required.

Unfortunately, things are not as simple as they seem. Recall that **substituted steps** involve backtracking and throwing away certain steps that have already been proposed. These steps may have had misconceptions associated with them, in which case the score for the path should no longer be influenced by these misconceptions. So the total score for the path may *increase* after a **substituted step** has been found, which violates the monotonicity requirement.

5.6 The solution

It turns out that heuristic scores *can* be used to guide a best-first search after all, if the search space is viewed in a slightly different way.

Consider again the example input path of

$$a \rightarrow \cdots \rightarrow b \rightarrow \cdots \rightarrow c \rightarrow \cdots \rightarrow d \rightarrow \cdots \rightarrow e$$

Rather than starting off by finding a connection between e and d, start off with *two* options:

- find a connection between e and d, or
- assume that d is an incorrect step (sort of a "red herring") and find a connection between e and c ignoring d.

The second option will have a lower score, since it ignores a mentioned step when we have no compelling reason to do so, so it will not be explored unless the score for the first option becomes sufficiently low.

So Pathfinder will try to connect e to d. If it succeeds, it will produce an expanded path with an associated score. If it fails, it will produce a degenerate path with e marked as an **extra step**. Either way, the result is a scored path; depending on the number and severity of the misconceptions identified in this segment of the path, the score may be higher or lower than the score for the "red herring" path. Note that this eliminates the problem of distinguishing between **extra** and **substituted steps**; the former are proposed as a last resort when no path can be found between two steps, and the latter are proposed for *every* mentioned step (although they will rarely be extended).

At this point, there are three options:

- find a connection between d and c
- assume c is an incorrect step and try to connect d to b

- assume d is an incorrect step and try to connect e to c (the remaining option from the previous iteration)

Again, only the most likely of these will be chosen. Note that there is no longer any backtracking, so scores will remain monotonic. Any point where the old version would have backtracked and skipped a step is already present as one of the "red herring" paths. It is true that in order to avoid backtracking we have to add additional possible paths at each mentioned node. But since best-first search can now be used, these paths will never be explored unless all of the already expanded paths are less likely, so there is no additional work involved in adding these paths.

There is still one technical point that needs to be addressed to guarantee monotonicity. Although backtracking has been eliminated, **substituted steps** will still cause previous scores to be changed. This is because the "red herring" score is only an approximation; the actual score cannot be computed until the corresponding correct step has been located and compared with the user's incorrect step. This is not a problem as long as the original estimated score for a **substituted step** is guaranteed to be greater than or equal to the actual score that will replace it later.

6 Implementation

The path expander and misconception identification modules of Pathfinder have been implemented in Common Lisp, and run on a Sun Workstation. Pathfinder is currently connected to a small database of plans about the tax domain, and can successfully handle examples from all five simplified categories of misconceptions, including several variations of the examples from the sample dialogue.

7 Conclusions

User misconceptions are unavoidable when interacting with people, and plan recognition systems for natural language interfaces need the ability to handle mistakes made by the user. Previous research in plan recognition has either ignored the problem entirely, used methods that rely on impossible assumptions about the knowledge of the system, or made unacceptable restrictions on the types of plans that can be handled. We have shown that misconceptions can be detected in a much larger range of plans than before without depending on methods such as "bug libraries".

Plan-based misconceptions can be categorized into fifteen classes, and these classes can occur virtually anywhere in the plan hierarchy. But the situation is much simpler than it appears at first glance. These categories can be simplified into five detectable classes, and in the case of isolated queries it is only necessary to check for these misconceptions in a very restricted section of the plan hierarchy.

The five simplified categories are not without problems, however. One consequence of **substituted steps** is that many different sets of misconceptions can be proposed to explain a single user query. The correct explanation must be chosen from these alternatives in order to provide the user with a helpful correction for his mistakes. Misconception scores can be

used to reduce the search space and to choose among the multiple possible paths by guiding a best-first search of the plan hierarchy.

There are several areas in which additional research is needed; some of these are: relaxing the isolated query assumption, handling **ordering** misconceptions, finding the appropriate path after a **substituted step** has been detected, and generating appropriate scores.

8 Appendix I: A demonstration

The following is an edited version of an actual transcript of Pathfinder's response to the second question in the example dialogue:

"I'm trying to fill out my taxes. Where do I get the W2 form for my fellowship?"

The input to the system is the semantic representation labeled "original query". Since the scoring algorithms are not yet implemented, the various possible answers are simply listed in approximate order of the number of misconceptions that they contain. Line breaks have been changed to improve readability.

Note that one "obvious" path, adding an extra **get** step to **fill-in-brown-fellowship**, is not present. This is because the current method for handling substituted and extra steps does not always check all the appropriate possibilities. This will hopefully be corrected soon.

Once the scoring algorithms are added, only the most likely plan will be presented; the others will not even be fully expanded, which will save a significant amount of time.

```
----------------------------------------
Original query is:

((DO-TAXES) (FILL-IN-FELLOWSHIP (FELLOWSHIP-OF *YOU))
            (GET 'FORM-W2 ?WHERE *HERE))

The user wants to do his taxes.
In doing so, he wants to fill in the information for fellowship FELL235.
In doing so, he wants to get FORM-W2, which is at ?WHERE,
and bring it to HERE.

There are 5 possible plans that the user could have.
--------------------
The first possible plan is:

Start with the user's original goal to do his taxes.

One way to do his taxes is to do his taxes himself.

One way to do his taxes himself is to do the 1040 form himself.

In order to do the 1040 form himself, he must fill in the 1040 form.

In order to fill in the 1040 form, he must fill in line 21 of the 1040
form.

In order to fill in line 21 of the 1040 form, he must fill in the
information for fellowship FELL235.
```

One way to fill in the information for fellowship FELL235 is to fill in the information for general fellowship FELL235.

ERROR: The previous step has a violated precondition: It is not true that FELL235 does not come from Brown University.

In order to fill in the information for general fellowship FELL235, he must get FORM-W2, which is at BROWN, and bring it to HERE.

ERROR: The previous step has a violated binding: The user tried to use FORM-W2 where he should have used FORM-1099-MISC.

ERROR: The previous step has a violated precondition: It is not true that FORM-W2 is at BROWN.

The second possible plan is:

Start with the user's original goal to do his taxes.

One way to do his taxes is to do his taxes himself.

One way to do his taxes himself is to do the 1040 form himself.

In order to do the 1040 form himself, he must fill in the 1040 form.

In order to fill in the 1040 form, he must fill in line 7 of the 1040 form.

ERROR: The user tried to fill in the information for fellowship FELL235 when he should fill in the information for wages FELL235.

In order to fill in line 7 of the 1040 form, he must fill in the information for fellowship FELL235.

One way to fill in the information for fellowship FELL235 is to fill in the information for general fellowship FELL235.

ERROR: The previous step has a violated precondition: It is not true that FELL235 does not come from Brown University.

In order to fill in the information for general fellowship FELL235, he must get FORM-W2, which is at BROWN, and bring it to HERE.

ERROR: The previous step has a violated binding: The user tried to use FORM-W2 where he should have used FORM-1099-MISC.

```
ERROR: The previous step has a violated precondition: It is not true
that FORM-W2 is at BROWN.

--------------------
The third possible plan is:

Start with the user's original goal to do his taxes.

One way to do his taxes is to do his taxes himself.

One way to do his taxes himself is to do the 1040 form himself.

In order to do the 1040 form himself, he must fill in the 1040 form.

In order to fill in the 1040 form, he must fill in line 8 of the 1040
form.

ERROR: The user tried to fill in the information for fellowship FELL235
when he should fill in the information for interest FELL235.

In order to fill in line 8 of the 1040 form, he must fill in the
information for fellowship FELL235.

One way to fill in the information for fellowship FELL235 is to fill in
the information for general fellowship FELL235.

ERROR: The previous step has a violated precondition: It is not true
that FELL235 does not come from Brown University.

In order to fill in the information for general fellowship FELL235, he
must get FORM-W2, which is at BROWN, and bring it to HERE.

ERROR: The previous step has a violated binding: The user tried to use
FORM-W2 where he should have used FORM-1099-MISC.

ERROR: The previous step has a violated precondition: It is not true
that FORM-W2 is at BROWN.

--------------------
The fourth possible plan is:

Start with the user's original goal to do his taxes.

One way to do his taxes is to do his taxes himself.

ERROR: The user tried to fill in the information for fellowship FELL235
```

when he should do the 1040 form himself.

In order to do his taxes himself, he must fill in the information for fellowship FELL235.

etc.

The fifth possible plan is:

Start with the user's original goal to do his taxes.

ERROR: The user tried to fill in the information for fellowship FELL235 when he should do his taxes with a tax service.

In order to do his taxes, he must fill in the information for fellowship FELL235.

etc.

References

[AP80] James F. Allen and C. Raymond Perrault. Analyzing intention in utterances. *Artificial Intelligence*, 15(1):143–178, 1980.

[BB78] John Seely Brown and Richard R. Burton. Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, 2:155–192, 1978.

[Car86] Sandra Carberry. User models: The problem of disparity. In *COLING-86*, pages 29–34, 1986.

[Car88] Sandra Carberry. Modeling the user's plans and goals. *Computational Linguistics*, 14(3), 1988.

[Cha85] Eugene Charniak. Knowledge representation and the plan-recognition-as-planning^{-1} theory. Technical Report CS-85-01, Brown University, 1985.

[Cha88] Eugene Charniak. Motivation analysis, abductive unification, and nonmonotonic equality. *Artificial Intelligence*, 34:275–295, 1988.

[GC88] Robert Goldman and Eugene Charniak. A probabilistic ATMS for plan recognition. In *AAAI-88*, 1988.

[Gen79] Michael R. Genesereth. The role of plans in automated consultation. In *IJCAI-79*, pages 311–319, 1979.

[Joh85] William Lewis Johnson. *Intention-Based Diagnosis of Errors in Novice Programs.* PhD thesis, Yale University, 1985.

[JWW84a] Aravind Joshi, Bonnie Webber, and Ralph M. Weischedel. Living up to expectations: Computing expert responses. In *AAAI-84*, pages 169–175, 1984.

[JWW84b] Aravind Joshi, Bonnie Webber, and Ralph M. Weischedel. Preventing false inferences. In *ACL/COLING-84*, pages 134–138, 1984.

[Kap82] S. Jerrold Kaplan. Cooperative responses from a portable natural language query system. *Artificial Intelligence*, 19:165–187, 1982.

[Kau87] Henry A. Kautz. *A Formal Theory of Plan Recognition.* PhD thesis, University of Rochester, 1987.

[May80] Eric Mays. Failures in natural language systems: Applications to data base query systems. In *AAAI-80*, pages 327–330, 1980.

[McC86] Kathleen F. McCoy. The ROMPER system: Responding to object-related misconceptions using perspective. In *ACL-86*, pages 97–105, 1986.

[McC89] Kathleen F. McCoy. Generating context sensitive responses to object-related misconceptions. *AI Journal*, Forthcoming, 1989.

[Pol86] Martha E. Pollack. *Inferring Domain Plans in Question-Answering.* PhD thesis, University of Pennsylvania, 1986.

[QDF88] Alex Quilici, Michael G. Dyer, and Margot Flowers. Recognizing and responding to plan-oriented misconceptions. Technical Report UCLA-AI-88-1, UCLA, 1988.

[SD73] Charles F. Schmidt and John D'Addamio. A model of the common-sense theory of intention and personal causation. In *IJCAI-73*, pages 465–471, 1973.

[SI81] Candace L. Sidner and David J. Israel. Recognizing intended meaning and speakers' plans. In *IJCAI-81*, pages 203–208, 1981.

[SSG78] C. F. Schmidt, N. S. Sridharan, and J. L. Goodson. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence*, 11:45–83, 1978.

[SSP85] James C. Spohrer, Elliot Soloway, and Edgar Pope. A goal/plan analysis of buggy pascal programs. Technical Report YALEU/CSD/RR#392, Yale University, 1985.

[vB87] Peter van Beek. A model for generating better explanations. In *ACL-87*, pages 215–220, 1987.