Classifying and Detecting Plan-Based Misconceptions for Robust Plan Recognition

Randall J. Calistri

Ph.D. Dissertation

Technical Report No. CS-90-11
May 1990

Classifying and Detecting Plan-Based Misconceptions for Robust Plan Recognition

by
Randall J. Calistri
B.S., SUNY-Albany, 1986
M.S., SUNY-Albany, 1987
Sc.M., Brown University, 1989

Thesis
Submitted in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy in the Department of Computer Science
at Brown University.

May 1990

Vita

I was born in Syracuse, New York on September 2, 1966. Growing up in a very musical family (my parents Tom and Bev Calistri were both music majors, and my grandfather had his own big band for 40 years), there was never any doubt that my first love would be music. I started playing the cello in fourth grade, and continued without interruption until last fall, when it had to be put on hold during the final thesis crunch. I started getting interested in musical composition in junior high school, at the same time that I encountered my second love: foreign languages. I began with Russian, and switched to German after moving to a new school without a Russian teacher.

I discovered my third love when I was bitten by the computer bug in high school. I convinced my parents that a computer was absolutely essential to our survival, and we ended up with a Franklin Ace 1000, one of the first legitimate Apple clones. That was the beginning of several years of happy hacking.

I entered SUNY-Albany in 1983, with all intentions of double-majoring in computer science and music. But as flexible as the programs were, I discovered that the only way to accomplish this would be to give up my languages, which I was not willing to do. So I settled for a double major in computer science and applied math instead, and concentrated my musical studies in a composition minor. That allowed me to continue with German and Russian, and to spend a remarkable summer studying at the University of Braunschweig. I finished my BS (summa cum laude) in 1986, and decided to stay one more year for a master's degree. It was then that I became fascinated with artificial intelligence. I was introduced to Paul Jacobs at General Electric R&D, and convinced him to be an outside advisor for my master's project, which was a natural language interface for operating systems.

I decided to come to Brown University for my PhD, and started working with Eugene Charniak in the fall of 1987. Fairly quickly, we zeroed in on a fascinating

project which combined natural language, plan recognition, and that peculiar tendency for people to make mistakes. The project rapidly grew into my thesis topic, which culminated in this dissertation. Thanks to Paul, I was able to return to GE as a summer intern in 1988 and 1989, so my research benefited from both the intense scholarly environment of academia and the real-world pragmatism of industry.

I am now anxiously looking forward to rejoining my fourth and greatest love (who has been around as long as the second), my fiancée Millie Yeh. Since her instrumental and linguistic interests are just as keen as mine, we are anticipating many long years of music-making and traveling to far-off lands. Once I lock down a research position at an as yet undisclosed location in upstate New York, all that remains is to dust off my poor neglected cello, buy some more manuscript paper, and update my passport.

Acknowledgments

As with all major undertakings, this work was not done in a vacuum. There are many people who have helped, both directly and indirectly, over the years.

My advisor, Eugene Charniak, has contributed a wealth of knowledge and expertise, and helped me struggle through some very nasty problems. He has given me the freedom to explore in my own way, and knows the fine art of how to guide without holding me back. He has also supplied most of my financial support at Brown, courtesy of the National Science Foundation and the Office of Naval Research. My committee members, Tom Dean and Jeff Vitter, have provided many helpful suggestions and were able to maintain an impressive turnaround time on the chapters. I would also like to thank my unofficial outside readers, Martha Pollack and Paul Jacobs. Martha's own dissertation laid the foundation for much of the current work in plan-based misconceptions, and her comments on my research have been very valuable. Paul is probably the reason I got into AI in the first place; he stuck by me through a master's thesis at SUNY Albany and two summers at GE, and has opened up many opportunities for me.

During my graduate years, I have had the rare pleasure of working in two exciting and stimulating environments. The AI group at Brown provided many invigorating discussions which were extremely helpful in critiquing earlier papers and talks. Special mention goes to Glenn Carroll, Robert Goldman, and Solomon Shimony of the WIMP group for getting me interested in combining natural language and plan recognition. My officemates, Bob Cohen and Sridhar Ramaswamy, had to put up with an occasionally irrational person who constantly monopolized the computer, and they are to be commended for their patience, good humor, and moral support. The people in the AI Program at General Electric R&D supported me through two summers, and gave me a very rich environment for learning how things are done in the real world. They were more than willing to include me in their group, and always had the time to answer my

questions.

I owe more than I can possibly acknowledge to my parents, Tom and Bev Calistri. From my first days, they have been a constant source of encouragement, and they never doubted for a moment that I would succeed. The love, support, and dedication that they have shown to me over the years is something for which I will always be grateful. I am truly honored that they volunteered to do the final proofreading for this dissertation.

This work is dedicated to my fiancée Millie Yeh. She put up with all the impossibilities of a long-distance relationship, and provided me with the best possible motivation for getting out in record time.

Abstract

In order to maintain a truly cooperative dialogue, an intelligent natural language interface must be able to recognize misconceptions in the user's plans. This involves solving two problems: determining what sorts of mistakes people make when they reason about plans, and figuring out how to recognize these mistakes when they occur.

There are sixteen distinct ways that a plan can be improperly constructed. Each of these corresponds to one type of plan-based misconception. But it is not necessary to consider all sixteen classes; the classification can be simplified to ten categories of detectable, distinguishable plan-based misconceptions. This classification can be used to develop a robust plan recognition algorithm, which is capable of recognizing both correct plans and plans that contain misconceptions. Probabilistic methods can be used to handle the three main difficulties of robust plan recognition: dealing with the "hard" classes of misconceptions, controlling the combinatorial explosion that results from the ambiguity inherent in faulty plans, and selecting the most likely plan among multiple competing explanations.

The theory that is developed here is implemented in a program called Pathfinder, which is a probability-based plan recognition system based on the **A*** best-first search algorithm that has been extended to recognize novel plan failures in an intelligent interface environment.

Contents

List of Figures

List of Examples

Chapter 1

Introduction

Abstract: *This thesis extends traditional methods of plan recognition to handle situations in which agents have flawed plans. This involves solving two problems: determining what sorts of mistakes people make when they reason about plans, and figuring out how to recognize these mistakes when they occur.*

Pathfinder is a probability-based plan recognition system based on the **A*** *best-first search algorithm that has been extended to recognize novel plan failures in an intelligent interface environment. It uses a new classification of plan-based misconceptions which categorizes all ways that a plan can fail.*

1.1 Why Worry About Misconceptions?

The field of plan recognition has been firmly established in artificial intelligence for almost 20 years, and researchers have been making progress on the problem of identifying the plan that an agent is following based on various types of external observations. Interestingly, the overwhelming majority of these researchers have made one of two assumptions about the agent and his[1] plans: either that the agent always has correct plans, or that the agent may make one of a small number of known mistakes in his plan.

[1] In keeping with a growing trend in the plan recognition community, this paper refers to the person carrying out the plan (user, actor, agent, questioner, speaker) as "he", and refers to the person recognizing the plan (human expert, observer, answerer, listener) as "she", regardless of the actual gender of the participants.

This seems like a reasonable approach to take, for several reasons. Admittedly, people occasionally do make mistakes. But most people have correct plans most of the time, so a large majority of the plans that we will actually see are going to be valid in the first place. And when people do make mistakes, they often make common mistakes that could be predicted ahead of time. Thus, given the types of simplifying assumptions that AI systems typically make about the world, the assumption that people have valid plans seems quite benign.

So why isn't this approach good enough? Why do we need to worry about plan-based misconceptions at all? It turns out that mistakes are much more frequent and varied than originally thought. 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. This last case is particularly serious because the agent is not even aware that he is making a mistake.

Most plan recognition systems assume that the agent will have perfect plans, and sometimes this is acceptable. But intelligent interfaces are specifically designed to interact with people, who may make mistakes. And in fact, it is the least experienced people, the people who make the most mistakes, who need intelligent interfaces the most. The more mistakes a person is likely to make, the more important it is to be able to recognize and correct these mistakes. This is in direct conflict with the assumption that we don't need to worry about mistakes.

1.2 An Example

To illustrate this type of situation, consider a hypothetical expert system in the domain of income taxes. It knows everything there is to know about filling out tax forms. Some of this information is displayed in Figure 1.1, which shows a portion of the hierarchy of plans used in filling out taxes.

The system has an intelligent natural language interface, where users can enter their questions in English and get back English responses. This system would be a cross between the TeleTax system, which is an automated tax assistance program but only has pre-recorded messages accessible through a menu, and the tax hotline, which can answer novel questions in English but requires human experts to be available during

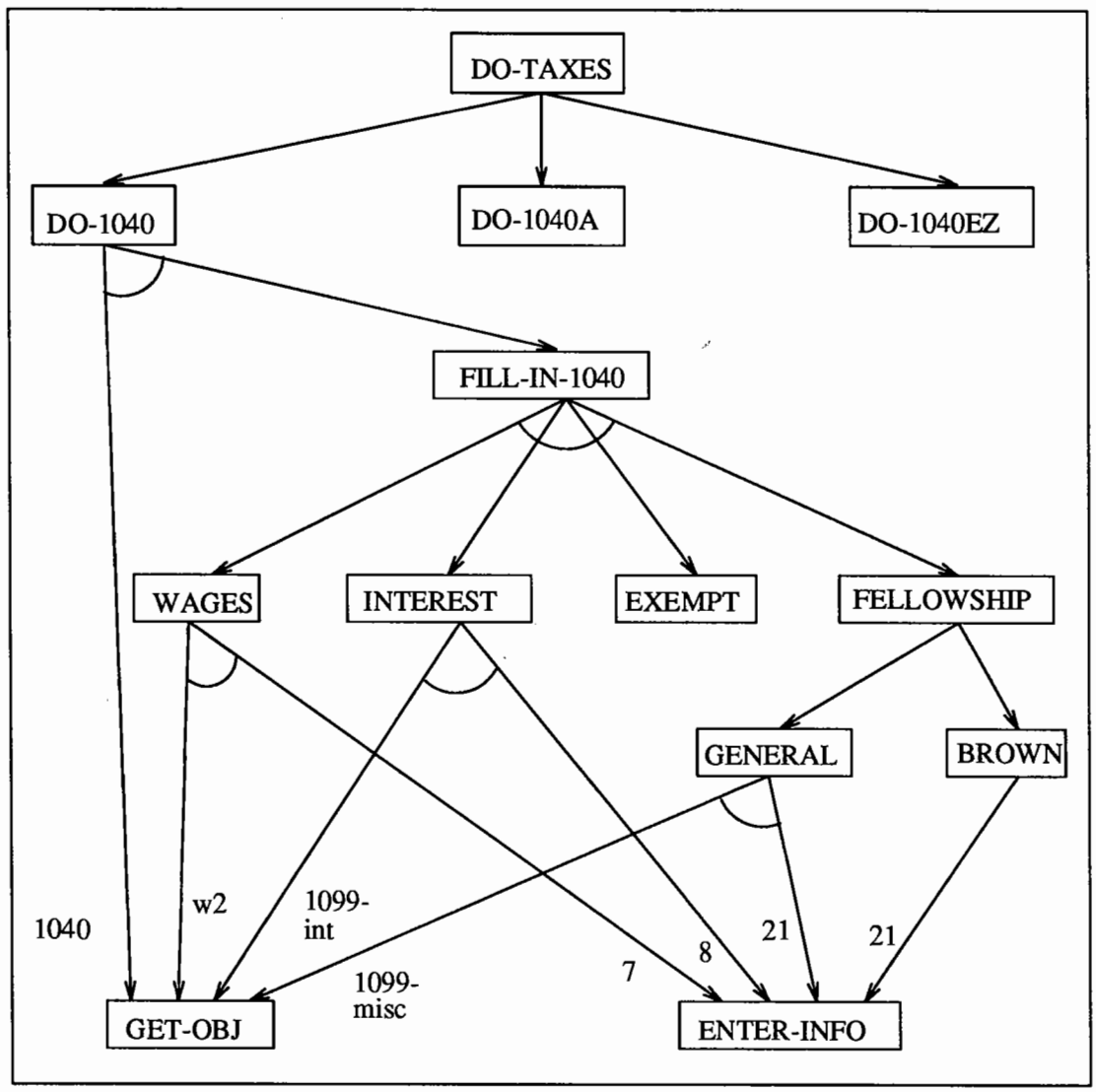

Figure 1.1: The plan hierarchy for the income tax domain.

its working hours.

Now assume that there is a graduate student (GS) from Brown University that is using this expert system (ES) to find out how to report the income from his fellowship, as in Examples 1 and 2:

> [GS] I'm trying to fill out my 1040 income tax form. Where should I list the exemption for my graduate fellowship?
>
> [ES] Fellowships are not exempt any more. You should list your fellowship on line 21, but exempt income should not be listed at all.

Example 1: The graduate student and his fellowship, part 1.

> [GS] All right, where do I get the W2 form for my fellowship?
>
> [ES] First of all, you want a 1099-MISC form, not a W2 form. And secondly, Brown University doesn't give out any 1099 forms this year.

Example 2: The graduate student and his fellowship, part 2.

At first glance, there does not seem to be anything mysterious happening in this example. Granted, the student is having some difficulty figuring out what to do with his fellowship, but the expert system realizes what is wrong and corrects him. In fact, this is exactly the behavior that we would expect from a human expert.

But let us look at this example more carefully. In the first exchange, the student is trying to fill out his 1040 form by listing exempt income on some unknown line, and is using his fellowship for that exempt income. The expert system cannot simply try to retrieve an appropriate plan for this, because there *is* no plan for entering exempt income information, and even if there were such a plan it would not accept a graduate fellowship as exempt income. In order to recognize the student's plan, the expert system must realize that fellowships could be mistaken for exempt income (because they actually *were* exempt until the 1986 tax reform), and that a student might try to report exempt income (perhaps because all other types of income are reported).

The second part of the example is even more complex. There are actually several

flawed plans that the student could have. He might simply be confused between 1099 forms and W2 forms (both are tax forms that one receives for income, and W2 forms are much more common), and he might not realize that the university doesn't give them out (in fact, Brown only stopped giving them out in 1988). Or he may know that most fellowships get 1099 forms, but he thinks that Brown gives out W2 forms instead. Or he may think that fellowships are a type of earned income (like teaching assistantships and research assistantships, two other common types of student income), which actually *do* get W2 forms. None of these plans would be stored in the expert system's library of correct plans, and it is very unlikely that the system would have been programmed with knowledge about all of these specific mistakes. So they would have to be dynamically constructed during the plan recognition process.

It is important to realize that plan recognition is critical for this example. We cannot just blindly answer the question that was asked, because the answer would either be vacuous or misleading. For example, a direct answer to the first question does not exist at all. And the only possible direct answer to the second question would have to ignore the phrase "for my fellowship", and would mistakenly send the graduate student off to some office for a form that he isn't even supposed to have.

We also need to note that the student's plan in this dialogue is ambiguous; we are not exactly sure what his actual plan is in the second query. Ambiguity already shows up quite often in the traditional plan recognition problem, irrespective of misconceptions. But flawed plans are *inherently* ambiguous, and the number of alternative explanations is much larger than with correct plans. Without careful precautions we will become swamped with competing explanations for the agent's actions.

Finally, this example illustrates that misconceptions do not always occur individually; quite often a single plan can have several misconceptions (which may be related to each other or may be completely distinct) that can conceal what the agent is attempting to do even further.

We are now almost diametrically opposite the original plan recognition assumption; rather than being able to ignore mistakes, we are now completely overwhelmed by them. 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 plan that the user actually has in mind, can seem remote.

1.3 Overview of the Approach

Luckily, things are not as hopeless as they appear. While it is true that we cannot possibly know all misconceptions that an agent might have in his plan, it is possible to identify all *types* of plan-based misconceptions. Rather than treating misconceptions simply as isolated beliefs, they can be classified according to how they affect the agent's plan.

The approach that we propose for reasoning about plan-based misconceptions consists of three major parts:

1. Developing a classification of plan-based misconceptions,
2. Developing and implementing an algorithm for plan recognition that can handle these plan-based misconceptions in a reasonable way, and
3. Specifying a probabilistic interpretation of robust plan recognition that can be used to guide a best-first search through the plan space and to select the most likely plan from among several competing explanations.

The Classification

By analyzing the structure of a plan, we will see that there are sixteen distinct ways that a plan can be improperly constructed. Plans consist of four components: bindings, preconditions, steps, and temporal orderings. Each of these components can fail in four ways: a component can be **violated** (the component is present, but has not been satisfied properly), **substituted** (the correct component has been replaced by an incorrect version), **missing** (the component should be in the plan but isn't), or **extra** (the component should not be part of the plan at all). Each of these sixteen combinations corresponds to one type of plan-based misconception.

But even though these misconceptions may occur anywhere in the plan hierarchy, it is not necessary to search the entire hierarchy, and it is not even necessary to consider all sixteen classes. Because of the restricted amount of information that is available from a dialogue, we only need to search for misconceptions in the part of the hierarchy that is directly related to the user's query. And because an outside observer does not have access to the actual beliefs inside the user's head, there are certain categories of misconceptions that are undetectable. Other categories can be combined because there is no way for the observer to distinguish between them. This results in a simplified

classification of ten types of plan-based misconceptions that are both detectable and distinguishable to an outside observer.

This classification can be used to develop a robust plan recognition algorithm, which is capable of recognizing both correct plans and plans that contain misconceptions. Although misconceptions could be treated as rather ephemeral beliefs abstracted from plans, this would not solve the problem of recognizing the plans that contain these misconceptions. Instead, by associating misconceptions directly with plan components, we impose a structure on the mistakes that allows us to handle misconceptions and plan recognition simultaneously.

The Program

The traditional plan recognition problem can be posed as a search problem, where the object is to find a path in the plan hierarchy that explains the user's query and is consistent with what we know about the world. This can be handled by a best-first search algorithm such as **A***. However, some modifications are needed to handle the various classes of plan-based misconceptions. Misconceptions can be divided into two large groups, based on how they affect the **A*** algorithm. **Constraint** misconceptions are quite easy to handle because they do not affect the structure of the user's plan. His plan can still be found in the plan hierarchy, but there may be certain violations that need to be explained. **Structural** misconceptions, on the other hand, are much harder because the user has incorrect beliefs about how the steps of a plan are put together. In these cases, the user's actual plan will not be found in the correct plan hierarchy at all, and we need to modify the structure of the hierarchy itself to identify the plan.

Two relatively simple extensions to the **A*** algorithm allow us to handle situations where the user has flawed plans. The extension for constraint misconceptions does not affect the asymptotic complexity of the algorithm at all, so plans with these misconceptions can be recognized just as efficiently as correct plans. But the extension to handle structural misconceptions results in a very severe combinatorial explosion. These misconceptions are fundamentally more difficult to handle, and even with substantially greater effort we cannot guarantee that we have properly understood the user's query. But there are heuristics that can identify the structural misconceptions that actually occur in practice, and that can be used to control some of the explosion.

The Probabilities

Probabilistic methods can be used to handle the three main difficulties of robust plan recognition: dealing with the "hard" classes of misconceptions, controlling the combinatorial explosion that results from the ambiguity inherent in faulty plans, and selecting the most likely plan among multiple competing explanations. Probabilities provide a sound theoretical basis for judging whether a particular plan or misconception is "likely". We can show that with certain reasonable assumptions, the probability of a particular plan can be calculated incrementally from the probabilities of the steps and misconceptions that make up the plan.

The probability of the user choosing a particular step in a plan is fairly easy to get. But the probability of the user having a particular misconception seems much more difficult, especially if we have no way of anticipating the misconceptions ahead of time. Luckily, there is a way around this. It is possible to break the probability of a misconception down into a number of independent "features", which can be calculated at runtime when the misconception is identified. Each class of misconceptions has associated with it a set of functions for calculating the relevant features. So once a misconception is identified and classified, we know how to calculate its features and combine them to get the probability of the misconception.

We can use the probabilities to form a heuristic function suitable for use with a best-first search. Given a potential explanation for the user's query, we can combine the probabilities of each step and misconception in the explanation to form the probability that this is the user's actual plan. This can be calculated incrementally by factoring in the steps and misconceptions as they are proposed by the algorithm. So as the best-first algorithm constructs an explanation, we can use the probability of the partial explanation constructed so far as a heuristic estimate of the likelihood of the entire explanation. This heuristic is appropriate for guiding the best-first search, as well as for ranking the final competing explanations for the user's query.

1.4 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 the theory and the practicality of the program, I have compiled a large corpus of naturally occurring dialogues, comprising well over 250 pages of text. This corpus includes over 100 exchanges that contain potential examples of plan-based misconceptions [17]. 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 deals with income taxes; it focuses on federal taxes, but also has some questions about state and local taxes. The "experts" who answer the questions in this setting are self-proclaimed, and there are frequently cases where several experts disagree with a certain diagnosis. 28 examples were extracted from this source.

- Transcripts of the radio talk show "Harry Gross: Speaking about Your Money", broadcast on February 1–5, 1982 by radio station WCAU in Philadelphia[2]. 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

[2]Thanks to Martha Pollack and Robert Kass for providing these transcripts.

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 was often trying 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 jumped to a conclusion too soon. 35 examples were extracted from these transcripts.

- 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 Pennsylvania[3]. 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. 12 examples were taken from these transcripts.

- 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. There are 25 examples, extracted from the following papers: [47, 67, 69, 83, 85, 110].

- Miscellaneous examples from the income tax domain that were collected during personal conversations with various people. There are 11 examples in this category.

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 supplied all of the examples that are used in this thesis. They have also served to validate both the developing theory and the Pathfinder program. A further analysis of the examples will be given in Chapters 4 and 5, showing how well the theory predicts the examples and how well the program can handle them.

[3]Thanks to Ethel Schuster for providing these transcripts.

1.5 Overview of the Thesis

Chapter 2 reviews the previous work that has been done in plan recognition and misconceptions, and sketches the relationship of plan-based misconceptions to several other research areas such as intelligent interfaces, intelligent tutoring systems, automatic program debugging, story understanding, and natural language response generation.

Chapter 3 provides formal definitions of plans and plan-based misconceptions. It also provides a formal statement of the particular problem that this thesis addresses, shows how a robust plan recognition module fits into a full intelligent interface system, and makes explicit the assumptions about the input and output of the system.

Chapter 4 proposes a complete classification of plan-based misconceptions, shows how the classification can be simplified, and illustrates it with examples of actual dialogues that involve flawed plans.

Chapter 5 describes how a traditional approach to plan recognition based on the **A*** heuristic search algorithm can be modified to deal with various types of plan-based misconceptions. It also presents the Pathfinder program, and gives a complexity analysis of the algorithm.

Chapter 6 gives a detailed description of the use of probabilities as search heuristics, and of the particular probabilities used in Pathfinder.

Chapter 7 summarizes the contributions of this thesis and points out several remaining areas of future research.

Chapter 2

Previous Work

Abstract: *Although a substantial amount of research has been done in the two separate areas of plan recognition and misconceptions, surprisingly little has been done in the overlapping area of plan-based misconceptions. After reviewing the relevant work from these two areas and showing the relationship of plan-based misconceptions to several connected research areas, this chapter describes the main systems that have been developed for detecting plan-based misconceptions.*

The study of plan-based misconceptions can basically be viewed as the intersection of the fields of plan recognition and misconceptions. While each of these fields has had a long history of productive research, there is surprisingly little overlap. However, research in plan-based misconceptions is strongly connected to a number of research areas which have traditionally been considered separate topics, including story understanding, intelligent tutoring systems, response generation, program verification, and intelligent assistants. Some of these areas can lend insight into how plan-based misconceptions can be studied, and others are areas which could benefit from knowledge of these misconceptions, but are currently lacking the ability to identify them. This is illustrated in Figure 2.1[1].

[1]This figure is only meant to show how the various topics are interrelated. The size of each region and the amount of overlap is not significant.

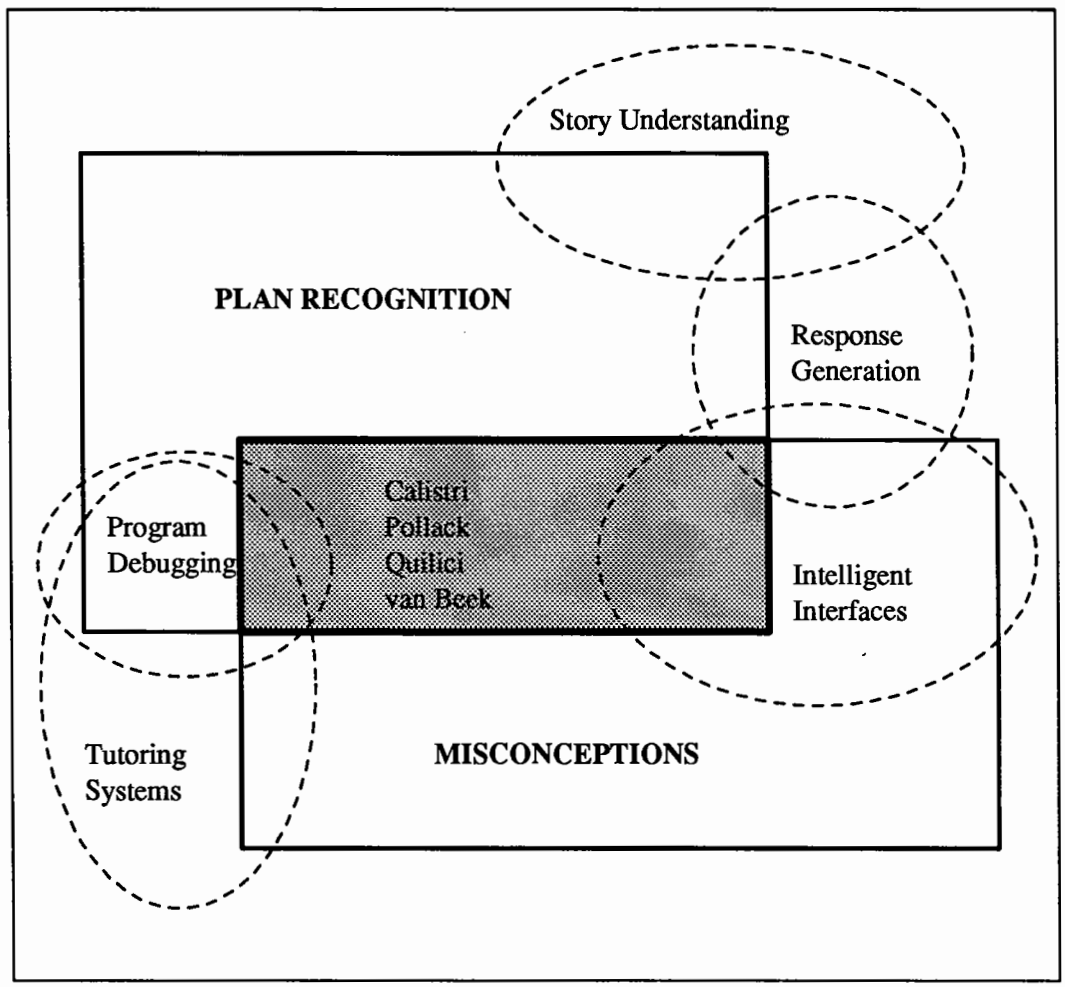

Figure 2.1: Plan-based misconceptions and related areas.

2.1 Plan Recognition

The origin of research in plan recognition is often traced back to Schmidt's 1978 paper [94], since this is the first paper to actually use the phrase "plan recognition". Here, he proposes a formal definition of the plan recognition problem:

> "The problem of plan recognition is to take as input a sequence of actions performed by an actor and to infer the goal pursued by the actor and also to organize the action sequence in terms of a plan structure" (p. 52).

But Schmidt actually formulated the basic problem of plan recognition at least five years earlier. His 1973 paper [95] sets the goal of creating "a computer-implemented model which will accept descriptions of actions and provide as output the reasons why the persons acted in the manner described." Already at this point, most of the fundamental problems of plan recognition have been recognized: context dependency, ambiguity, combinatorial explosion, and even the possibility of faulty plans.

But for the next 17 years, the problem of faulty plans has been almost completely ignored; most researchers in plan recognition either never address the problem at all [28, 30, 53, 54, 61, 94], or only mention it in passing [3, 20, 37, 60, 95].

2.1.1 Intelligent Tutoring Systems

One area that must explicitly reason about an agent's faulty plans is intelligent tutoring systems. Here, the primary goal is to identify and correct a student's misconceptions in such a way that he can learn from his mistakes. It is here that we would expect to see the most work on plan-based misconceptions.

Indeed, there is almost universal acceptance that plan recognition is critical to the process of intelligent tutoring, and that a student's misconceptions play an important role in this recognition. The dominating method for handling the misconceptions has been the **bug list** [6, 7, 15, 16, 36, 38, 39, 102, 106]. In this method, the standard set of correct plans for the domain is augmented by another set of *incorrect* plans (the bug list) that the student may be following. This is a very useful method for catching the most common mistakes; it effectively "pre-compiles" certain patterns of errors that we would expect to see quite frequently, and it is often much more efficient than other methods of detecting misconceptions. The bugs are used to form a model of the student's knowledge; this can be structured as a "differential model", specifying how

the student's knowledge differs from the correct domain knowledge, or as an "overlay model", which encodes the student's knowledge as a proper subset of the system's knowledge of correct and incorrect plans [104].

However, while they are useful for augmenting other detection methods, bug lists are quite unsatisfactory by themselves. They require that the designers of the system must be able to predict all mistakes that the student could make. This becomes completely intractable immediately; people are capable of a potentially infinite variety of mistakes, and there is no way to predict all of them ahead of time.

Brown and Burton analyzed the very simple task of basic subtraction, and found 60 primitive bugs that could be combined to form 330 different mistakes [15]. Even with these 330 "buggy" plans, they are only able to account for 39% of the actual mistakes that students made. Burton [16] later extended this classification to 110 primitive bugs, and found that almost 40% of the diagnoses involved more than one bug; some had up to 4 bugs in a single subtraction problem. Of course, it is notoriously difficult to impose a more sophisticated plan structure on tasks as basic as arithmetic, since the student's plan can frequently deviate wildly from the designer's correct plan. For these types of problems, empirical bug lists are often the best method available.

Besides the difficulty of identifying the bugs in the first place, another critical problem with using bug lists as a general solution is that they are extremely domain-specific. The bugs correspond to specific mistakes that individual users make on particular problems, and a completely new bug list must be built for different domains. For example, Brown's subtraction system can't even generalize to addition [5].

If bug lists perform so poorly in a domain as simple as subtraction, how can we hope to encode all possible mistakes for a more complicated domain such as income taxes? One possibility is to group bugs together into more general categories and try to synthesize the bugs as they are needed, rather than storing all of them ahead of time. For example, Matz [65] proposes that students form bugs by incorrectly extrapolating from a known correct rule. He mentions various types of extrapolation that account for 30 common types of errors, but these are still very domain-specific.

Sleeman [103] suggests two alternatives: in one, the system reasons both forward and backward until it cannot go any further, then "invents" a new rule to fill in the gap in the user's plan. In the other, the bug list is replaced by a set of perturbations that can be applied to the correct domain rules.

2.1.2 Program Debugging

Automated program debugging is a field closely related to intelligent tutoring, where the goal is to automate the process of debugging computer programs. While many of the implemented program debugging systems still rely primarily on bug lists (for example, [86]), researchers are beginning to realize that such an approach is unreasonable for large systems, since it amounts to "compiling" all the tutorial knowledge possessed by the system [62].

An alternative approach is to base debugging on the semantics of the program. For example, Murray [72] parses the student's algorithm into a semantic form, and does a heuristic search on each of the functions in the algorithm; if no correct function is found, he selects the "closest match" based on semantic similarity. The student's misconceptions correspond to the differing semantic features of the incorrect functions.

Reiser et al [87, 88] do essentially the same thing, but they emphasize providing immediate feedback: their system is interactive, and as soon as an incorrect step is detected, they report the closest semantic match. This immediate correction helps them simplify the problem of compounding errors.

Johnson [46] emphasizes **intention-based diagnosis**, where he tries to infer the reason why the student included each line of the program. He allows the use of bug lists, but when no known program plans match the student's input, he finds the closest match and uses plan-difference rules which try to explain the differences. The rules that are actually used in his PROUST system tend to be very specific (for example, there is a rule that is used only when unexpected lines are found inside an "if" statement), and while they are able to abstract away from individual bugs, they are still tied to specific programming constructs and would not easily generalize to other domains. However, the final classification of bugs that Johnson provides is a bit more domain-independent; it breaks plans up into seven different components, and accounts for four types of failures for each component.

Spohrer, Soloway and Pope [105] argue that this classification is in fact *too* independent; they suggest that emphasis should not be on just the failed plan components, but also on the context in which the components appear. I will argue that we need precisely this sort of contextual information, but that it does not belong in the classification.

2.1.3 Discourse Analysis and Story Understanding

The people that have arguably made the most use of plan recognition have come from the natural language community. It has long been recognized that understanding a story requires reasoning about the plan that the actor is following [1, 24, 40, 43, 114]. In addition, if the actor's plan contains misconceptions, it is quite likely that the reader will be unable to understand certain aspects of the story unless she can recognize the flaws in his plan. However, story understanders invariably assume that all of the actors will have rational, correct plans. Things may happen in the story that cause these plans to fail, but the plans themselves are initially sound.

Participating in a dialogue also requires following the speaker's goals and plans, both to understand what he is saying [20, 28, 60, 99] and to respond appropriately [3, 48, 74]. This work is more directly related to misconceptions. In particular, Joshi, Webber, and Weischedel [48] treat the problem of how to phrase a response in such a way as to correct any known misconceptions and to avoid introducing any new ones. However, all of these systems assume that the misconceptions have already been identified by some other module.

2.1.4 Intelligent Interfaces

One particular application of plan recognition that has been receiving more attention recently is the work on intelligent interfaces. These systems are designed to make the user's job easier by assisting him with certain tasks, anticipating problems that might arise, allowing more natural communication, and providing a true "do-what-I-mean" environment.

Intelligent interfaces often take the form of natural language interfaces [27, 115, 116] or graphical interfaces [42]. They often have the features of both "assistants", which help the user with his routine tasks, and "advisors", which help the user when he runs into problems. These advising systems differ from tutoring systems; with a tutoring system, the tutor is in control and can present lessons or examples of its own choosing. An advising system, on the other hand, must be prepared to handle novel problems that the *user* presents [70].

Again, many of the intelligent interface systems rely heavily on bug lists to handle user errors. Carroll [22] undertook an interesting experiment which demonstrated the ineffectiveness of this method. He simulated an intelligent help system for a database

program, where hidden people typed in the messages that the "system" was supposed to be generating. He started out with a pre-compiled list of bugs and responses, but rapidly discovered that the majority of messages had to be generated on the fly. His conclusion was that "people are incredibly creative at generating errors and misconceptions, and incredibly fast." Interviews with the subjects later revealed that in many cases they expected the system to "accurately discern their *intentions*, regardless of how indeterminately these were reflected in their actions." These two results argue extremely forcefully for robust plan recognition that can handle novel plan-based misconceptions.

2.2 Misconceptions

The formal study of misconceptions, or of errors in general, has a history even longer than that of plan recognition. It is also much more interdisciplinary; Singleton [100], for example, relates errors and error classification to a wide range of disciplines including ergonomics, psychoanalysis, cybernetics, decision theory, and social theories.

2.2.1 Classifying Misconceptions

Some of the early studies of errors focused on unintentional or careless mistakes rather than true misconceptions, and the researchers were primarily looking for ways to improve the reliability of data entry personnel and computer operators. Bailey [12] argues that there is nothing inherently error-prone in people, and that it is possible to identify and control the factors that cause people to make mistakes in the hopes of eliminating (or at least reducing) these mistakes. His classification of error sources includes factors such as training, environment, and documentation.

Nawrocki [73] proposed a method for classifying data entry errors that has become quite popular in AI: collect data from the domain in question, manually locate all of the errors, and create a group of mutually exclusive categories to classify the errors. The problem with this approach is that it is unprincipled and leads to ad hoc, domain-dependent categories that are not easily extensible and are not guaranteed to account for future errors.

Van Eekhout [111] proposed a more independent classification for the types of errors that troubleshooters make when diagnosing equipment failures. These are actual misconceptions rather than just careless mistakes, and the diagnosis procedures closely

resemble plan structures. His classification focuses on plan components, but at a very high level; the categories related to plans are "choice of goal", "choice of procedure", and "execution of procedure", and each of these can be either incomplete, inappropriate, or completely missing.

Rouse and Rouse [91] claimed that such an independent method is insufficient, and that a useful classification needs to have an additional level of specific categories that are more task specific. Their most significant change is to augment the "execution of procedure" category with several additional categories, resulting in a classification that is quite similar to the classification of plan-based misconceptions presented in Chapter 4.

2.2.2 Responding to Misconceptions

Kaplan [50] and Mays [66] dealt with how to respond to a question that contains a failed **presupposition** (where there is no direct answer to the question) or **presumption** (where there is exactly one direct answer to the question). Such situations usually signify the presence of a misconception, because in cooperative conversation people are expected to ask questions only when there is a choice of answers. They show that these simple types of misconceptions can be detected by checking the referents of objects that are mentioned in the query to see whether any of them are vacuous.

Webber [113] extended the work on presuppositions by breaking them into two cases: misconceptions about what *is* the case, and misconceptions about what *can be* the case. Her point is that certain beliefs could be considered "deviant" because it is impossible for them to ever be true, and that other beliefs, while currently incorrect, may be true in another state of the world.

McCoy [68, 69] has been working on the problem of correcting object-related misconceptions. These are incorrect beliefs that an agent has about objects in the world. She distinguishes between two types of object-related misconceptions: **misattributions**, where an object is given an attribute/value pair that it does not possess, and **misclassifications**, where an object is placed in the object taxonomy incorrectly. Correcting these misconceptions involves three steps: denying the incorrect belief, stating the correct information, and justifying the denial and correction. She presents a small number of correction strategies based on the classification of a misconception.

However, with the exception of Kaplan and Mays, none of the classification or response systems provide any way of detecting misconceptions. And most of the misconception work has concentrated on non-plan-based misconceptions. The classification systems provide organizational categories for the misconceptions, and the discourse systems are able to handle the misconceptions once they are identified. But what has been conspicuously missing is a method for detecting the misconceptions in the first place.

2.3 Plan-Based Misconceptions

There are only a few systems that provide a full treatment of plan-based misconceptions; such a system must deal explicitly with misconceptions that affect an agent's plans, must provide a classification for organizing these misconceptions, and must suggest ways to detect them when they occur.

2.3.1 Pollack's SPIRIT System

The first researcher to take a serious look at the problem of plan-based misconceptions from both a classification and detection perspective was Pollack [83]. Her claim is that intelligent agents not only need to recognize the fact that another agent's plan is invalid, but also need to identify the incorrect beliefs that cause it to be invalid.

She treats plans as mental phenomena, and views "having a plan" as having a certain configuration of beliefs and intentions. An observer judges a speaker's plan to be invalid when there are discrepancies between the beliefs that the observer ascribes to the speaker as a result of having this plan, and the beliefs that the observer has herself.

The specific cases of plan-based misconceptions that Pollack addresses are instances where the speaker has a **simple plan** and is asking about a particular step in that plan. Simple plans 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.

The SPIRIT system works by taking the speaker's goal and queried step, and trying to construct an **explanatory plan** (EPLAN) to connect the two using a set of plan inference rules. This involves ascribing certain beliefs to the speaker, such as a belief

that one step generates another, or a belief that it is possible to execute a certain step in the current state of the world. Any incorrect beliefs that are encountered during the construction of an EPLAN are plan-based misconceptions.

Pollack distinguishes between three types of plan-based misconceptions: **unexecutable acts**, which involve incorrect beliefs about executability, **ill-formed plans**, which involve incorrect beliefs about generation, and **incoherent queries**, where the observer is unable to form any EPLAN at all that explains the speaker's query.

This work lays the foundation for all of the subsequent research in plan-based misconceptions; it presents a classification for how simple plans can fail, and gives a method for detecting these types of failures. The main drawback to this approach is its restriction to simple plans; most interesting plans are not "simple" at all, and it is not obvious how this theory could be extended to deal with more complex plans.

Another problem, one that is shared with *all* of the current methods, is the inability to handle multiple interpretations of plans. Even "perfect" plan recognition (with no misconceptions) is frequently ambiguous; adding incorrect beliefs only serves to greatly increase the level of ambiguity. None of the current systems even address the issue of choosing among competing interpretations.

2.3.2 Quilici's AQUA System

Another system that deals directly with the problem of plan-based misconceptions is the AQUA system of Quilici, Dyer, and Flowers [85]. Given a set of incorrect user beliefs, they try to infer other missing or mistaken user beliefs that might have led to these incorrect beliefs. The system first tries to find explanations for why it does *not* hold the user's belief, then it tries to explain why the user *does* hold the belief.

They distinguish between three types of user beliefs that can fail, producing plan-based misconceptions: beliefs about **plan applicability conditions** (whether a particular plan should be used to achieve a goal), beliefs about **enablement** (whether a particular state must exist before a plan can achieve a goal), and beliefs about **effects** (whether a state will exist as a result of a plan's execution). For each of these types of beliefs, the system has a set of potential explanations for why an agent would or would not believe it.

Quilici's work basically continues where this thesis leaves off: it starts off with an initial top-level incorrect belief (a plan-based misconception), and attempts to find

the underlying incorrect beliefs that caused it, and will collect the information necessary to generate a more helpful correction. This distinction between "top-level" and "underlying" misconceptions is discussed in greater detail on page 38.

2.3.3 Van Beek's System

Van Beek [110] has extended and implemented Joshi's earlier work[48, 49], and so his system also addresses the problem of generating appropriate responses to mistakes and misconceptions. While Joshi only considered the user's immediate stated and intended goals, van Beek argues that it is necessary to address other domain and background goals that the user may have, and if these goals conflict with the immediate goals it indicates a misconception on the part of the user.

The method used here is based on a theorem prover. The system first determines whether the stated goal is possible. If it isn't, it informs the user and gives the cause of the failure. If there is some alternate plan that will achieve the stated goal and is compatible with the higher level goals, this alternative is presented.

Assuming that the stated goal is possible, the system determines whether it really does help achieve the intended goal. If it does not, the system informs the user and tries to find an alternative stated goal. Even if the stated goal is appropriate here, it checks to see if there is any other goal that would be more optimal. This is done by running an embedded planning system to generate an optimal plan for the intended goal. Of all the papers mentioned in this chapter, this is the only one that addresses the problem of how to diagnose suboptimal plans.

2.3.4 Other Approaches

Broverman and Croft [14] are currently designing an interactive planner for use in a cooperative office environment. As it is cooperative, it must be able to recognize other agents' plans in addition to generating its own plans. The planner is *not* viewed as an omnipotent agent with complete knowledge of all plans, so when it discovers that the user is executing an unknown plan, it could either be because the user has a misconception, or because he is using a plan that the system is simply unaware of. It therefore tries to negotiate with the user until both agree on a single correct plan.

Sebrechts [97] combines two methods to create a different approach to the problem of plan-based misconceptions. He starts with "intention-based diagnosis", which uses

normal plan recognition with bug lists, and then adds "associative networks", which use a neural network approach to activate the most likely goal based on various information from the user's query.

Ohlsson and Langley [76, 77] take a psychological approach to **cognitive diagnosis**, which is closely related to plan recognition. They start off with a particular domain problem, a problem space, a procedure for generating the correct solution to the problem, and the agent's proposed solution to the problem. They then produce a solution path in the given space that leads from the problem to the answer, calculated by performing a best-first search through the subject's problem space. Their main restrictions are that there must be a known way to generate correct answers, that the problem space must include ways to generate all possible incorrect answers, and that the solution path that is produced often does not have an obvious interpretation.

2.4 Summary and Comparison

The overall objective of this thesis, to be able to handle robust plan recognition when the agent has flawed plans, is identical to Pollack's. But both the classification and approach that I take are substantially different. My classification for plan-based misconceptions is probably closest to Rouse's [91] and Johnson's (as described in [105]), but it is not tied to the domain of computer programs as Johnson's is.

The approach that I take is to reason directly about the structures of the agent's plans, rather than about his individual beliefs. This is quite different from most of the plan-based misconception approaches discussed here, but is somewhat similar to Johnson's and Murray's idea of analyzing the semantics of the student's functions while remaining within the framework of the program structure.

A major claim of this thesis is that there is a large ambiguity problem with flawed plans; once an agent is allowed to have invalid plans, there are often many competing explanations for the actual plan that he is following. While most of the work discussed here has ignored this problem, Charniak and Goldman [40, 41] advocate using probabilistic reasoning to select among competing explanations for story understanding. I use a similar (but much simpler) probabilistic approach both for ranking the possible explanations and for guiding a heuristic search through the space of possible plans.

Chapter 3

Basic Definitions

Abstract: *There are several different ways of posing the plan recognition problem, and there are several different assumptions that can be made about the observer and the agent participating in the planning process. This chapter presents formal definitions for the plan recognition problem and the various assumptions. It also presents a formal semantics for the structure of plans, and defines what it means to have a plan-based misconception.*

3.1 The Plan Recognition Problem

At the most abstract level, the plan recognition problem can be thought of as something like the following:

> *Given certain observations about the agent's actions and knowledge about the current and past states of the world, identify the plan that the agent is currently following.*

However, this definition is underconstrained in two ways: it does not specify what types of observations are allowed, and it does not specify what it means to "identify a plan".

The types of observations that are allowable are usually determined by the domain. For example, in a natural language system, the observations are utterances of the speaker [3]. In math systems, they may be the individual digits of the solution [15]. In intelligent assistants for aircraft pilots, they may be instrument readings and control panel inputs [9, 90]. These observations may all be atomic (at the finest level of detail), they may all be at a more abstract level, or they may be at varying levels of granularity.

The meaning of "plan identification" is more dependent on the type of plan recognition that the observer wishes to do. It might involve finding the lowest level task that incorporates all of the atomic observations. It could involve tracing a single observation up through more and more abstract plans until some "top-level" goal is reached (this is sometimes referred to as **intent inferencing**, since it focuses on the intentions of the agent rather than the particular details of the plan [18, 37]). Or it could involve expanding all the details of the plan, possibly at several levels of granularity, effectively reconstructing the complete plan that the agent is currently following.

The plan recognition problem in this thesis addresses a variation of intent inferencing. Specifically, we start with one specific plan step that has been extracted from the user's query. The task is to trace up from this mentioned step to a top-level goal, identifying all of the intermediate plans along the way, but not necessarily performing an explicit expansion of the intermediate plans. This vertical path, which connects the top-level goal to the specific queried step, is not true intent inferencing, since the intent (top-level goal) is often already known. But it does constitute plan recognition, because the object is to discover the role that the mentioned step plays in the overall plan that the user is currently following.

3.2 Common Assumptions in Plan Recognition

There are several assumptions that are frequently made to simplify the task of plan recognition. These involve assumptions about both the agent's and the observer's beliefs and their knowledge about plans in the domain. Not all of these assumptions are present in all approaches, and the assumptions that are present are not always made explicit.

Perhaps the most common assumption is the **Closed World Assumption** (CWA), which basically states that the observer is a true expert in the domain, and knows all correct ways of achieving the goals in the domain.

Assumption 1 (CWA) *The observer has complete and correct knowledge of all valid domain plans.*

This is as much an assumption about the domain as it is about the observer; there are many domains for which this assumption is completely invalid, regardless of the

observer's level of expertise. However, there are also many domains where the Closed World Assumption can be reasonably approximated.

But this says nothing about the agent's knowledge of plans, or about how the observer's plan set compares to the agent's plan set. To relate these two sets of plans, there are two variations on the Closed World Assumption. If it is applied to invalid plans instead of valid plans, we are assuming that the observer is familiar with all possible mistakes that the agent could make. This is called the **Bug List Assumption** (BLA):

Assumption 2 (BLA) *The observer has complete and correct knowledge of all possible invalid plans that the agent may have.*

And if we make the Closed World Assumption with respect to the agent instead of the observer, we get the **Perfect Plan Assumption** (PPA):

Assumption 3 (PPA) *The agent has complete and correct knowledge of all valid domain plans.*

One final assumption is the **Rational Plan Assumption** (RPA), which says that the agent does not knowingly and intentionally make any mistakes.

Assumption 4 (RPA) *The agent always selects an appropriate plan, relative to his beliefs about plans and the state of the world.*

Various combinations of these assumptions place various demands on the capabilities of the agent and the observer, as shown in Figure 3.1. The one combination of assumptions in the figure that is not applicable is all four assumptions occurring at the same time. If the agent always selects a correct plan, then there is no need for the observer to have a bug list.

The Rational Plan Assumption states that the agent acts rationally relative to his view of the world. If this is combined with the Perfect Plan Assumption, it states that the agent acts rationally relative to the *observer's* view of the world. It would be unusual to assume perfect plans without rationality; we would not expect to find an agent who has complete and correct knowledge about valid plans, but who ignores this information in his selection of an appropriate plan.

Almost all current systems make the Closed World Assumption, and most also require either bug lists or perfect plans. Both of these combinations mean that the plan being followed by the agent is always known to the observer. The difference

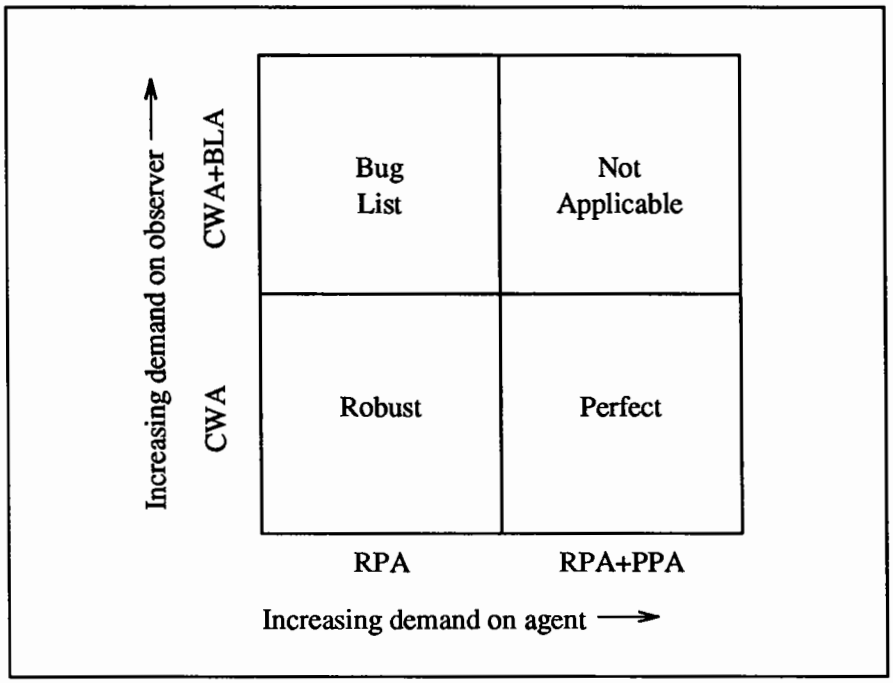

Figure 3.1: Assumptions in plan recognition.

between the two is that bug lists place the bulk of the responsibility on the observer, while insistence on perfect plans places more on the agent. These assumptions make the job of plan recognition much easier, since the observer never needs to worry about encountering unknown plans; the agent will always be following either a known valid plan or a known flawed plan.

The most restrictive combination is the Closed World Assumption, the Perfect Plan Assumption, and the Rational Plan Assumption (which is called "Perfect Plan Recognition"). In this case, the observer is guaranteed that the agent will be following a known, correct plan that is appropriate for the current circumstances, so she will never have to worry about any kind of plan-based misconception at all.

Although the Closed World Assumption is occasionally relaxed to allow the agent to have novel plans, it is much rarer to relax the bug list or perfect plan requirement. This thesis retains the Closed World Assumption for correct plans, but makes no assumptions at all about the observer's knowledge of incorrect domain plans. Furthermore, it does not require perfect plans; agents may be missing certain correct plans, and they may have additional incorrect plans.

Of course, it is impossible to avoid the Perfect Plan Assumption entirely; we need

to make *some* assumptions about the knowledge and actions of the agent. This thesis assumes that the agent is more or less rational; in other words, he has a particular plan of action (which may or may not be correct) that he is following, and that he is not consciously making any mistakes (i.e., the Rational Plan Assumption still holds). Further, the agent's set of plans is assumed to be approximately the same as the system's set of correct plans, *modulo* a relatively small number of misconceptions. This is not at all the same as the Bug List Assumption. It does *not* assume that these misconceptions can be predicted ahead of time. There are still an infinite number of misconceptions that the agent could potentially have, but the total number of these misconceptions that are actually in a particular plan in the agent's plan hierarchy is relatively small.

In fact, we make a slightly stronger assumption, which basically corresponds to a nonmonotonic (defeasible) version of the Perfect Plan Assumption: unless there is evidence to the contrary, we assume that the agent has exactly the same plan hierarchy that the observer does. In other words, there is a default assumption that the agent's plan hierarchy is correct and complete. If there is evidence that the agent has a flawed plan, this is revised to an assumption that the agent's plan hierarchy is correct and complete *except for this particular plan.* This corresponds to a weak assumption of global rationality; agents always behave rationally relative to their own views of the world, and they are assumed to behave rationally relative to the observer's views of the world *unless there is evidence to the contrary.*

Thus, the specific plan recognition problem that this thesis addresses (which is called "Robust Plan Recognition" to contrast it with "Perfect Plan Recognition") is the following:

> *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, as described above.

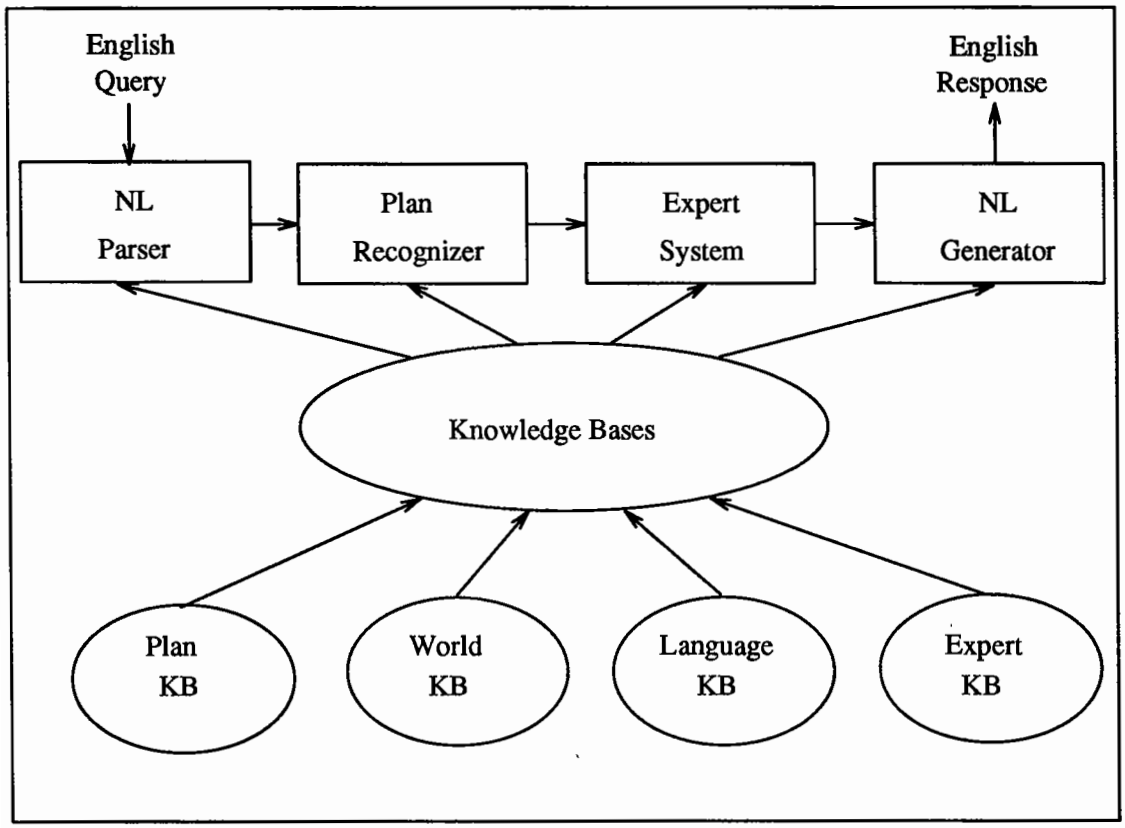

Figure 3.2: A complete intelligent interface system.

3.3 The Role of Plan Recognition in an Intelligent Interface

The approach to plan recognition that is developed in this thesis should be viewed as only one component of a larger intelligent natural language interface system. The purpose of the plan recognition module is to provide additional information to an expert system by placing the user's query within the context of a plan and by identifying any relevant misconceptions that the user may have.

The placement of the plan recognition module within the complete system is pictured in Figure 3.2. The user's query is parsed by a natural language understander, which has access to the plan hierarchy. This provides the plan recognizer with a semantic description of the specific plan step that the user is asking about, and might also be able to provide descriptions of some of the higher level goals. For instance, after processing the question about the W2 form in Example 2, the parser can extract

certain points in the plan hierarchy that it knows the student is considering. From the student's utterance, the parser knows that he is trying to get a W2 form, it knows that he is trying to fill in his fellowship information, and it knows that he is filling out his income taxes. So the parser may output something like:

`(do-taxes)` $\rightarrow \cdots \rightarrow$ `(fill-in-fellowship f235)` $\rightarrow \cdots \rightarrow$ `(get-obj w2-form)`.

Since these points in the hierarchy all describe the same plan at various levels of detail, they will all lie along the same vertical path in the hierarchy. Notice that the parser is always guaranteed to find such a path, since the minimum required input (a top-level goal and a queried step) is satisfied by the degenerate path

`(generic-plan)` $\rightarrow \cdots \rightarrow$ `(queried-step)`.

The information about these known points is passed to the plan recognizer, which attempts to discern the relationship between these plan fragments. In this case, by examining the plan hierarchy, the recognizer realizes that there is no valid plan involving getting a W2 form for a fellowship. It therefore concludes that there may be misconceptions present in the plan (for example, that the object in the `get-obj` step is not correct), and that there may be multiple possible interpretations of what the agent is attempting (as described on page 4). These interpretations will be ranked according to likelihood, and will be passed on to the expert system.

The expert system now has a great deal more information than just the original query. The query is set within the context of a plan, and misconceptions that would not be obvious from just a semantic interpretation of the isolated query have been identified as well. The expert system can now make whatever calculations are necessary to answer the query or to solve any problems that have arisen.

These calculations and solutions, along with the misconceptions, are passed along to the natural language generator, which decides how much of the information to include in the answer and actually generates the response. For example, it might decide that because of the user's misconception, an answer to his original query doesn't even make sense; it is more important to tell him that he doesn't want a W2 form than it is to tell him where the W2 forms are kept. In other cases, the misconception might be unimportant enough that it can safely be ignored, and the original question can be answered directly. In still other cases, the best response might contain both an answer to the original question and a correction of any misconceptions.

3.4 The Structure of Plans

Since this thesis takes a structure-based approach to misconception classification, it is necessary to define the structure of plans before the classification can be developed. While this makes the classification somewhat dependent on the representation of plans, this representation has been chosen to be quite flexible and to encompass the features of many of the common representations that are currently being used in other systems.

3.4.1 Plan Components

The plans that are considered in this thesis have the following four components:

- **Parameters** are placeholders for objects that are used by 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 parameters. By convention, variables 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 parameters of `?what`, `?from`, and `?to`, and might have bindings of `object84`, `current-loc273`, and `new-loc5` respectively.

- **Preconditions** are propositions about the world that must be true for the plan to be executed, and determine whether the plan should even be considered by the agent. These are similar to **applicability conditions** [21] or **constraints** [42] in the sense that the agent cannot attempt to satisfy the conditions himself. Preconditions are always beyond the control of the agent (such as a precondition that a fellowship cannot be from Brown University in order to get a 1099 form), and are often time-invariant (such as a precondition that the agent of some action must be animate). Any condition that the agent *can* control will be represented as a **substep**, since the agent must have a plan for achieving these conditions.

- **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. These steps may be atomic actions, or they may be complex subplans. In the tax domain, the substeps for filling in information about an agent's wages might consist of getting a W2 form from his employer,

reading the amount on the form, and entering that amount on line 7 of the 1040 form.

- **Alts** are alternate ways of doing a plan. For example, there are two ways that an agent can fill out his taxes: he can either do it himself, or he can have a tax service do it for him. If alts are present, then there must be no substeps (and vice versa).
- **Add** and **delete** are lists of propositions that are used to model atomic actions. If both substeps and alts are empty, then the plan is assumed to take place without further decomposition (no subplan is needed), and the add list and delete list update the current state of the world to reflect the effects of the action.

- **Temporal constraints** place certain restrictions on the timing of steps. There are two different types of temporal constraints:
  - **Orderings** are temporal constraints placed on the order of execution of substeps. These are vacuous for alt steps and atomic actions, and enforce a partial order on substeps. Orderings consist of a list of ordered pairs of substeps, where the first substep must be executed before the second. If all substeps are totally ordered, the constraints can be abbreviated to "`total`" rather than listing all of the obvious pairs.
  - **Mutex** are mutual exclusion constraints placed on the execution of alt steps. They partition the list of alt steps into subsets of mutually exclusive steps that cannot co-occur. For instance, in a plan for investing money, it is possible to invest in both stocks and bonds, so these alternatives are not mutually exclusive. In contrast, the alts for choosing an income tax form (1040, 1040A, or 1040EZ) *are* mutually exclusive; the agent can only submit one of them. If the entire set of alts is mutually exclusive, this can be abbreviated to "`total`".

Plans can be grouped together into a hierarchy that forms a directed *AND/OR* graph[1]. The leaves of the graph correspond to atomic actions. Substep plans (plans which have **substeps**) form the *AND* nodes of the graph ("you need to do A *and*

[1]The algorithm presented in Chapter 5 restricts this graph to be acyclic, but the structure and theory developed in this chapter do not require this restriction.

B *and* C"). Alt plans (plans consisting of **alt** steps) comprise the *OR* nodes of the graph ("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.

3.4.2 Some Examples of Plans

This section contains some examples of plans from the income-tax domain. They are part of the hierarchy that was presented in Figure 1.1. The first example is a plan to fill in fellowship information. There is a single parameter, which presumably represents the fellowship to be filled in. There are two preconditions: that this item really is a fellowship, and that the agent has income from this fellowship. Finally, there are two mutually exclusive ways to do this plan: either fill in a general fellowship, or fill in a Brown fellowship.

```
(def-plan
  :name fill-in-fellowship
  :parameters (?x)
  :preconditions ((isa ?x fellowship)
                  (has-income agent ?x))
  :alts ((fill-in-general-fellowship ?x)
         (fill-in-brown-fellowship ?x))
  :mutex total)
```

If we expand out `fill-in-general-fellowship`, we see that there is still a single parameter for the fellowship, but now there is an additional precondition that this fellowship cannot come from Brown University (if it does, then the agent should be following the `fill-in-brown-fellowship` plan). There are also three substeps that must be done in the following order: the agent must get a 1099 form from the university, he must read the amount on line 4, and he must enter that amount on line 21 of the 1040 form.

```
(def-plan
  :name fill-in-general-fellowship
```

```
:parameters (?fell)
:preconditions ((not (equal (university-of ?fell) brown)))
:substeps ((get-obj 1099-form (university-of ?fell) here)
           (read-amt ?amt line-4 1099-form)
           (enter-info ?amt line-21 1040-form))
:orderings total)
```

Finally, if we look at the `get-obj` plan, we notice that this is an atomic action. The only precondition is that the object must currently be at the from-location, and the add/delete lists update the current state of the world by changing the location of the object.

```
(def-plan
  :name get-obj
  :parameters (?obj ?from ?to)
  :preconditions ((at ?obj ?from))
  :add ((at ?what ?to))
  :delete ((at ?what ?from)))
```

3.4.3 Comparison to Other Representations

The representation developed in Section 3.4.1 contains the major features of most of the current plan representations. The basic structure of atomic actions is identical to the primitive plan representation developed in STRIPS [35].

The hierarchical nature of plans was first introduced in ABSTRIPS [92], and has been further developed and used in most current representations to decompose plans into simpler subtasks [26, 30, 31, 61, 117]. However, there are still a few systems that continue to use flat representations (for example, [23]). Most of the hierarchical systems only contain substep decomposition; only a handful of systems explicitly contain alt nodes in the representation itself [9, 37]. Kautz [53] has the equivalent of alt nodes, but he maintains two completely separate hierarchies: a decomposition hierarchy (for substeps) and an abstraction hierarchy (for alts).

Temporal constraints were added to plans in NOAH [93]. Prior to this, substeps of a plan were implicitly assumed to be totally ordered. Most current representations have at least a minimum temporal expressiveness to handle partial ordering of substeps [20, 31], while some have the expressiveness of a full temporal logic [53].

There are some features that are present in a few systems that are not part of this representation. These include procedural structures (looping, iteration, etc.) [26, 31], the ability to refer to objects independently of parameters [26], and the ability to express varying levels of importance ("concern") for preconditions [63, 64].

3.4.4 A Formal Semantics for Plans

This section presents a formal semantics for the plan structure that was defined in Section 3.4.1. It uses the following notation:

Π	=	set of all propositions
τ, τ_0, τ'	=	time points
$\mathcal{W}_\tau$	=	state of world at time τ; $\mathcal{W}_\tau \subset \Pi$
$\mathcal{P}$	=	set of all plans
P_i	=	particular plan = $(p_i, t_i, s_i, a_i, \alpha_i, \delta_i)$
		$P_i \in \mathcal{P}$
		$p_i, \alpha_i, \delta_i \subset \Pi$
		$s_i, a_i \subset \mathcal{P}$
		$t_i \subset s_i^2 \cup a_i^2$

The six components of a plan have semantic interpretations of preconditions, temporal constraints, substeps, alt steps, add list, and delete list respectively.

t_i is a list of ordered pairs of either substeps or alt steps, with the semantic interpretation that if the pair comes from s_i the first step in the pair must be executed before the second, and if the pair comes from a_i the two steps are mutually exclusive. In the latter case, the pairs define a partition of a_i such that if two steps are in the same pair they are in the same partition[2].

α_i and δ_i are lists of propositions that are added to and removed from $\mathcal{W}_\tau$ (the current state of the world) respectively when an atomic action is executed.

Individual components of a plan will have the same subscript as the plan; for example, s_i will always refer to the substeps of plan P_i. Alternatively, components can

[2]It is these partitions, not the pairs, that are actually used in the implementation.

also be referred to by functions of the same name; for example, the substeps of plan P_i can be referred to by $s(P_i)$.

There are further restrictions on the sextuple P_i, resulting in three specializations of plans:

- The set of **substep plans**, $\mathcal{P}_s$, contains plans which have substeps and which may have preconditions and temporal constraints, but which have no alt steps or add/delete lists.

$$P_i \in \mathcal{P}_s \iff P_i = (p_i, t_i, s_i, \emptyset, \emptyset, \emptyset) \wedge s_i \neq \emptyset$$

- The set of **alt plans**, $\mathcal{P}_a$, contains plans which have alt steps and which may have preconditions and temporal constraints, but which have no substeps or add/delete lists.

$$P_i \in \mathcal{P}_a \iff P_i = (p_i, t_i, \emptyset, a_i, \emptyset, \emptyset) \wedge a_i \neq \emptyset$$

- The set of **atomic plans**, $\mathcal{P}_\alpha$, contains plans which may have preconditions and add/delete lists, but which have no substeps, alt steps, or temporal constraints.

$$P_i \in \mathcal{P}_\alpha \iff P_i = (p_i, \emptyset, \emptyset, \emptyset, \alpha_i, \delta_i)$$

These three specializations are mutually exclusive and exhaustive; any plan P_i must belong to exactly one of the specializations.

The predicate **executable** is defined on plans as follows:

$$\text{executable}(P_i, \tau) \iff \begin{cases} P_i \in \mathcal{P} \\ p(P_i) \subset \mathcal{W}_\tau \\ [\exists P' : P_i \in s(P')] \implies \forall (x, P_i) \in \text{Tr}(t(P')) : \\ \qquad\qquad\qquad\qquad \exists \tau' : \text{executed}(x, \tau') \wedge \tau' \preceq \tau \end{cases}$$

where $\text{Tr}(t(P'))$ is the transitive closure of the temporal constraints t for the plan P'. This means that a plan P_i is capable of being executed at time τ exactly when all of its preconditions hold in the state of the world, and if P_i is a substep of a higher-level plan, all of its siblings that precede it in the partial ordering must have already been executed.

The predicate **executed** has three definitions, one for each specialization of plan:

- A **substep plan** is said to be executed at time τ iff it is executable at time τ_0 and all of its substeps have been executed between τ_0 and τ:

$$P_i \in \mathcal{P}_s \ \wedge \ \text{executed}(P_i, \tau) \iff \begin{cases} s(P_i) \neq \emptyset \\ \text{executable}(P_i, \tau_0) \\ \exists x \in s(P_i) : \text{executed}(x, \tau_0) \\ \exists x \in s(P_i) : \text{executed}(x, \tau) \\ \forall x \in s(P_i) : \exists \tau' : \text{executed}(x, \tau') \ \wedge \ \tau_0 \preceq \tau' \preceq \tau \end{cases}$$

- An **alt plan** is said to be executed at time τ iff it is executable and at least one of its alts has been executed at time τ:

$$P_i \in \mathcal{P}_a \ \wedge \ \text{executed}(P_i, \tau) \iff \begin{cases} a(P_i) \neq \emptyset \\ \text{executable}(P_i, \tau) \\ \exists x \in a(P_i) : \text{executed}(x, \tau) \end{cases}$$

- An **atomic plan** is said to be executed[3] at time τ iff it is executable and all of its add/delete effects are reflected in the state of the world at τ.

$$P_i \in \mathcal{P}_\alpha \ \wedge \ \text{executed}(P_i, \tau) \iff \begin{cases} s(P_i) = a(P_i) = \emptyset \\ \text{executable}(P_i, \tau) \\ \alpha(P_i) \subseteq \mathcal{W}_\tau \\ \delta(P_i) \cap \mathcal{W}_\tau = \emptyset \end{cases}$$

  The result of executing an atomic plan is that the state of the world is updated appropriately: $\mathcal{W}_\tau \longleftarrow (\mathcal{W}_\tau \cup \alpha(P_i)) - \delta(P_i)$.

Note that the plan structure defined in Section 3.4.1 has one extra component, namely **parameters**. This is because the plans that are stored in the plan hierarchy are really plan *schemas*. They contain a list of parameters which may be used as variables throughout the definition of the schema. A plan is invoked by providing the name of the plan and a list of values to bind to the parameters; all occurrences of the parameters in the schema are replaced by the corresponding values. The predicates **executable** and **executed** hold only for plan *instances*, not plan *schemas*.

This section has not added anything new to the plan structure; it has simply formalized the ideas that were presented in Section 3.4.1 and given precise definitions for the notions of plan executability and plan execution.

[3]Technically, this does not guarantee that P_i was executed precisely at τ; it could have actually been executed at time τ_0 prior to τ. But this does guarantee that there were no subsequent actions that changed any relevant aspects of the state of the world between τ_0 and τ, which is all that we need.

3.5 Misconceptions and Plan-Based Misconceptions

Before continuing, it is a good idea to clarify exactly what is meant by a plan-based misconception. The term **misconception** is defined as follows:

> *A misconception is any incorrect belief that is held by the agent.*

In other words, the agent believes that proposition p is true, when in fact it is false (or vice versa). The agent must actually believe that he is correct, so this does not include phenomena such as careless errors and slips of the tongue[4].

Plan-based misconceptions are defined as a specialization of this concept:

> *A plan-based misconception is a misconception about a component of the agent's plan.*

This misconception can involve the value of a component, the value of one part of a component, or even whether part of a component exists at all. Plan-based misconceptions intuitively correspond to incorrect beliefs that directly cause a plan to fail.

There are two reasons for restricting consideration to plan-based misconceptions in this thesis. The first is that the main goal of this thesis is to extend traditional plan recognition techniques to make them more robust. In order to reason about plan-based misconceptions, we have to reason about the agent's plan, and thus do plan recognition. So these are the interesting misconceptions to look at from the perspective of plan recognition.

The other reason for limiting our attention to plan-based misconceptions is that in some sense this restricts us to a "top-level" mistake. Any plan-based misconception may have underlying incorrect beliefs that caused it. These incorrect beliefs may have been caused by even deeper incorrect beliefs, and so on. 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.

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-obj` step of a plan to fill in fellowship information is a **plan-based misconception**, since the use of this incorrect form directly causes the student's plan to fail. The

[4] Although the theory does not include such phenomena, they can often be handled successfully by treating them as if they were misconceptions.

reason *why* the student switched them is another level of misconception that can be used to explain the first. There are potentially an infinite number of misconceptions that he 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.

Each of these can in turn be explained by many deeper misconceptions, for example that all forms contain the same information regardless of their number, and so on.

Of course, we might know something about this particular person 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 [85]. 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 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 belief himself, whatever it may be (the response "you're supposed to get a 1099 form, not a W2 form" would probably suffice to correct all four potential underlying misconceptions listed above).

Chapter 4

A Classification of Plan-Based Misconceptions

Abstract: *This chapter presents the classification of plan-based misconceptions. There are sixteen ways that plans can fail, based on the structure of plans developed in Chapter 3. But this classification can be simplified into ten detectable and distinguishable types of plan-based misconceptions.*

4.1 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 3.4.1 can contain misconceptions, and there are several different ways that each of them can fail.

For the purposes of classification, all three types of steps will be grouped together, since only one of them can be present in any given plan anyway. Also, this paper will refer to binding misconceptions rather than parameter misconceptions, associating the misconception with the *value* of the parameter rather than with the parameter itself. Finally, both orderings and mutex will be grouped together, again because only one of them can be present in any given plan. This gives four basic components of a plan that can fail: **bindings**, **preconditions**, **steps**, and **orderings**.

Each of these components can fail in any of four ways, which can be generalized as follows:

- A **violated** component occurs when the agent has the correct component, but it

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

Figure 4.1: A classification of plan-based misconceptions.

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 agent should have included this component in his plan but didn't.
- An **extra** component occurs when the agent includes this component in his plan but shouldn't have.

These 16 types of plan-based misconceptions are summarized in Figure 4.1[1]. The descriptions above are very oversimplified, and are not meant to be taken literally. The following sections give a more detailed description of each type of plan-based misconception, illustrating each one with a variation of the graduate student's dialogue in Example 2 wherever possible.

[1] **Violated Steps** are purposely replaced by **Violated Optimization**. This is discussed on page 43.

Binding Failures

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

- **Violated Binding:** 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.

- **Substituted Binding:** Even if the agent has the right values, he may try to bind them to the wrong parameters. For example, in attempting to **get** a certain object, he may reverse the source and destination.

- **Missing Binding:** A plan may actually have n parameters, but the agent 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.

- **Extra Binding:** A plan may have n parameters, but the agent may specify more than n. He may think that in the plan for filling out his fellowship information, he also has to specify how many years he has been a graduate student.

Precondition Failures

As with bindings, **preconditions** can also fail in four different ways.

- **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 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.

- **Substituted Precondition:** 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).

- **Missing Precondition**: 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").

- **Extra Precondition**: 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.

Step Failures

We will now try to break **step** failures into the same four classes.

- **Violated Step**: This is the only combination of component and failure that cannot occur. Because of the hierarchical representation of plans, steps are themselves plans. So a violated step is simply a violated plan; it will be recognized as one of the other types of plan failures. Therefore, this type of failure can be ignored.

- **Substituted Step**: 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`.

- **Missing Step**: The agent should have included this step in his plan, but didn't. Missing substeps and atomic actions (add and delete steps) are the simplest to understand; the agent wasn't aware that it was necessary to do this step in order to complete his plan. For example, he might not know that he has to fill in his fellowship information when he completes his 1040 form. Missing alt steps are a bit trickier. An example of this would be if the agent didn't know that there was any difference between general fellowships and Brown fellowships. He would only have a single plan for filling out fellowships, and so would not have any alt node to distinguish between the two possibilities.

- **Extra Step**: The agent is including a step that doesn't belong. These are exactly analogous to missing steps. Extra substeps and atomic actions are obvious. An extra alt step implies that the agent is trying to distinguish between plans that are actually identical (such as filling in general fellowships and filling in Cornell fellowships, which are done in exactly the same way).

There is one additional misconception type that is associated with steps, but that does not fall into the violated/substituted/missing/extra paradigm. 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.

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.

- **Violated Ordering**: The agent thinks (correctly) that step A has to be done before step B, but thinks (incorrectly) that step A has already been done.
- **Substituted Ordering**: The agent thinks that step A has to be done before step C, but it really has to be done before step B.
- **Missing Ordering**: The agent doesn't even know that step A has to come before step B.
- **Extra Ordering**: The agent thinks that step A also has to come before step C, but it doesn't.

4.2 Simplifying the Classification

In the classification that has just been presented, some misconception categories are not possible in certain situations. In other cases, misconceptions can exist but are undetectable. Finally, some misconceptions, although distinct within the mind of the

agent, are indistinguishable by an outside observer. This is caused by the fact that the listener does not have the opportunity to observe the agent's misconceptions directly; she can only observe the effects that these misconceptions have on his query. These restrictions on the location, detectability, and distinguishability of misconceptions can be used to simplify the classification. Impossible and undetectable categories can be completely eliminated, and indistinguishable categories can be collapsed together.

Section 4.2.1 will discuss which types of misconceptions are detectable at all, and will describe the circumstances under which they can be detected. Section 4.2.2 will examine the remaining categories to determine whether they can be distinguished by an outside observer. Each of the final simplified categories is illustrated with an example from the dialogue transcripts that were discussed in Section 1.4.

4.2.1 When are Misconceptions Detectable?

Although certain types of misconceptions have restrictions on where they can occur, most types can occur anywhere in the plan hierarchy. But 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 observer will not have any knowledge about misconceptions that occur in parts of the plan hierarchy that the user has not discussed.

To be more precise, the plan hierarchy can be divided into five separate sections as shown in Figure 4.2. Each section will be examined to determine which types of misconceptions can exist there, and to determine whether they have any hope of being detected. The following isolated user query from Example 2 will illustrate the dimensions of the sections:

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

Queried step

The **queried step** is the most specific step that can be extracted from the user's utterance. This is referred to as a queried step whether the utterance is actually a query or whether it is a statement or command. In the fellowship example, the queried step is `(get-obj w2-form ?where here)`.

The one type of misconception that usually cannot occur at a queried step is a missing step. Since the user specifically mentioned the queried step, it is normally not

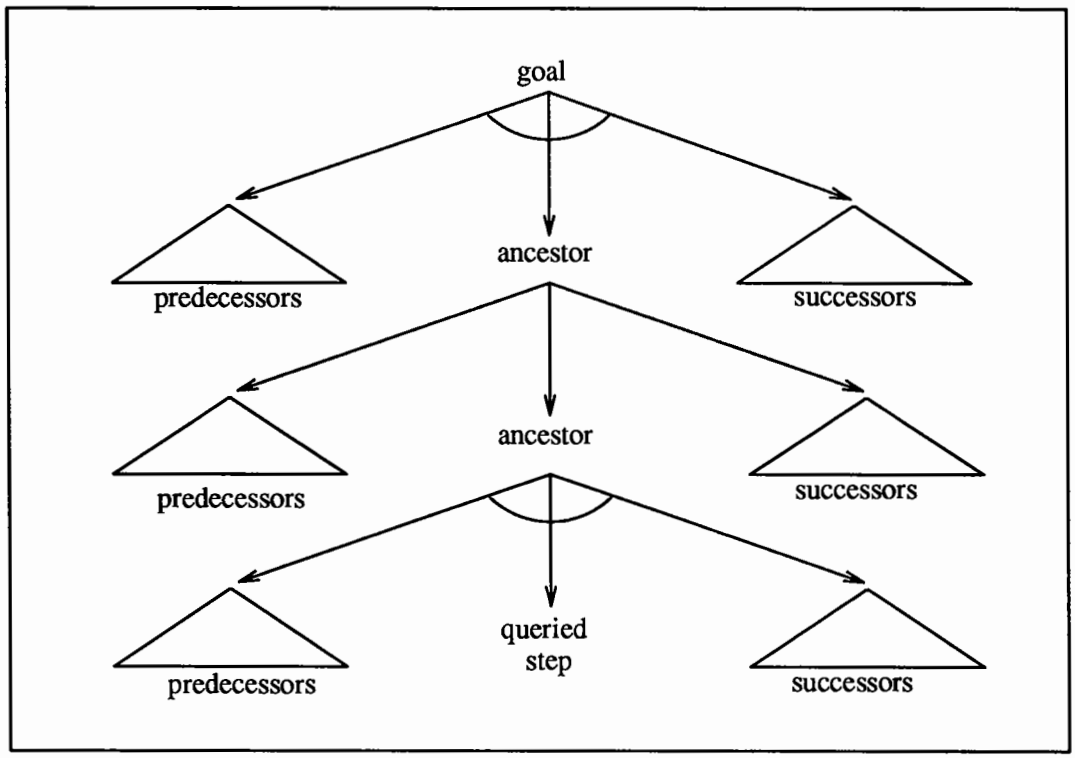

Figure 4.2: The search space.

possible for it to be missing (it may be wrong, but it has to be there). There are only two ways that a queried step can be missing. The first is if the agent is using the query to explicitly assert that the queried step is *not* part of the current plan, when in fact it is. The other way that it is possible to mention a "missing" step is if the user queries an entire procedure (i.e., a sequence of substeps) and omits one of the required steps.

Other than these two special cases, missing queried steps are impossible. But all other misconception types can occur at the queried step without restriction, and have a chance of being detected.

Goal

The **goal** is the most general step that can be extracted from the query. In this example, the goal could be `(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. Unlike the queried step, however, there is no special case where the query can be phrased in such a way that the goal is missing from the plan. So missing steps are completely impossible at the goal.

All misconception types other than missing steps are allowed and have a chance of being detected.

Ancestors

The **ancestors** are all nodes that lie along the vertical path between the goal (inclusive) and the queried step (exclusive). Note that this vertical path represents a single time period; each node in the path is either a substep of the node above it or an alternate way of doing the node above it. So in some sense, all nodes along the path represent the same plan step at different levels of detail. 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.

Once again, missing steps cause problems. There cannot be a missing step misconception at any of the *mentioned* ancestors, for the same reason that the goal 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 missing steps, all other misconception types can occur at ancestor nodes and have a chance of being detected.

Predecessors

The **predecessors** are all nodes that temporally precede the queried step. Assuming that steps are listed in chronological order from left to right, predecessors basically include all nodes to the left of the vertical path, such as getting the 1040 form. Steps that have no temporal order relative to the queried step are placed in the successors partition in order to restrict predecessors to steps that must have already taken place (again, relative to the queried step).

Any type of misconception can occur at a predecessor node, including missing steps, but they are all undetectable. There are two cases to consider, based on whether the node in question is part of a path that has been previously mentioned during the dialogue. In the case of a path that has already been discussed by the user, we can safely assume that the observer has already corrected all detectable misconceptions at the time that the path was discussed. In the second case, if the misconception is contained in a portion of the plan hierarchy that has not been discussed yet, it is undetectable. 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 observer will never be made aware of it.

There are some misconception types that we would expect to be detectable regardless of whether the user has mentioned them or not. For example, while some preconditions can change their truth value over time, others are time-invariant; they will always be either true or false. It should be possible to check if these time-invariant preconditions hold, whether or not they are in a portion of the hierarchy that has been discussed. We should also be able to check temporal constraints at any of the predecessors.

But it turns out that even these types of misconceptions are undetectable, because the observer may not even be able to determine whether a given node in the hierarchy is part of the user's current plan. Remember that a substantial percentage of the plan hierarchy consists of alt nodes, which act 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 observer has no way of knowing how the plan hierarchy has been pruned. This prevents the observer from checking preconditions in predecessor nodes (a misconception type that could normally be detected) because even if she finds one that has been violated, she doesn't know whether or not the user is going to try to execute that particular step. This step might be part of the hierarchy that he has already pruned off, so he would have no intention of trying to execute it, and a violated precondition should not be considered a misconception.

There is one exception where the observer *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. These steps are guaranteed to be part of (the correct version of) the user's current plan. Once the user selects a particular alt step for his plan, he is committed to selecting all of the substeps that comprise that plan. So time-invariant preconditions and temporal constraints can be checked for violations in these steps. However, we cannot make the same claim for the descendants of these steps; as soon as an alt step is encountered, we are back in the problem of the last paragraph.

With this single exception, all misconception types in predecessor nodes are undetectable.

Successors

The **successors** are all nodes that follow the queried step in time, and cover the right hand side of the plan hierarchy (including nodes that are not ordered with respect to the queried step).

Misconceptions located at successor nodes are analogous to the case with predecessors; there is often no way to tell whether a given successor node is part of the user's current plan or not. In addition, it is not possible to test temporal constraints of successor nodes at all (even in the special case mentioned for predecessors) because these are steps that will not be done until some time in the future. It may be that a currently unsatisfied condition may become satisfied before the step is executed, in which case no misconception has occurred. So misconceptions are not detectable in successors at all.

Summary

Although we were not able to completely eliminate any misconception classes, we were able to restrict the search space within which we can expect to locate the misconceptions. Both predecessors and successors can be ignored in our search for misconceptions, so the only section of the plan hierarchy that needs to be checked is the single vertical path from the goal to the queried step. Within this section, missing steps are only detectable at the queried step, and even that requires a special form of query.

4.2.2 When are Misconceptions Distinguishable?

This section examines each of the misconception categories to determine whether they are distinguishable from each other. After simplifying the classification by collapsing indistinguishable categories, each remaining type of plan-based misconception will be illustrated with an example from the transcripts.

Binding Categories

Bindings can be simplified into a single category.

Missing and **extra bindings** both occur when the agent has the wrong number of parameters for a particular plan. There is immediately a problem of logic representation that is caused by having an incorrect number of parameters. If plans are expressed in a logic such as first order predicate calculus, they get translated into functions, and it is not clear how to even represent a logic function with the wrong number of parameters.

But this will automatically be taken care of in the process of translating from the agent's English input to the internal representation of plan invocations. This translation is done by the natural language understander, which knows about the number of parameters that each plan requires. The parser will attempt to find values in the agent's query to bind to each of these parameters. Failing that, it will attempt to find some default value for the remaining parameters. Parameters that still have no value at this point must be missing from the agent's version of the plan, and will be filled in with Skolem constants ("placeholder" constants that have no known properties) [25]. In this way, the plan is guaranteed to have the correct number of parameters.

The question remains as to 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 for the plan. Missing bindings are also tricky. There are borderline examples such as investing in a CD on page 42. But it is unclear whether this is truly a missing binding or whether the agent is assuming a default value for the binding. If missing bindings do exist, they are identifiable by the fact that an instantiated plan still contains a Skolem constant (because the agent did not provide the necessary value and no default value could be found). An analysis of the examples in the transcripts did not find a single example of either a missing or extra binding. Given this lack of supporting evidence, missing and extra bindings will be eliminated from the classification.

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, in general it is 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 agent's bindings may have been incorrect even before the substitution, so even after the substitution is undone violations may still occur.

Again, no examples of substituted bindings were found in the transcripts. However, even if they do occur in real life, they are indistinguishable from violated bindings based on the analysis above, and the two categories can be collapsed together. Since we have already eliminated missing and extra bindings, this leaves **violated bindings** as the only type of detectable binding misconception.

As a simple example of a violated binding, consider the dialogue shown in Example 3 where a tax expert is explaining how to calculate estimated taxes[2]. The second expert's

[2]For all of the examples from the transcripts, all personal names have been eliminated, and have been replaced with [U] for the user (or novice, or person asking the question) and [E] for the expert

response shows that she believes the first expert's plan had a violated binding for the tax form that should be used.

[E1] ... Plugging $50,000 into Schedule Y-2 gives a tax due of $9976.50.

[E2] I think you meant Y-1.

Example 3: Violated Binding (easy).

While most cases of violated bindings are quite straightforward, they can occasionally get very messy. Example 4 is taken from the investment transcripts, and the entire dialogue appears to be derived from a single violated binding on the maturity date of a certificate of deposit. This one mistake generates several other violated bindings, and ends up complicating the situation to such an extent that even after several readings, the conversation is still not completely clear.

[E] If the penalty is going to be taken based upon the difference between the rate on the certificate and the passbook rate, that's gonna hurt you too much.

[U] Even though it's 4 more years?

[E] Well it isn't. It's only 2 more years.

[U] Oh 4 years — that's right, yes the other 3.

[E] Didn't you say 84?

[U] Right.

[E] Well that's 2 more years, maybe 2 1/2.

[U] Well this is 82, ... no I beg your pardon it's 86.

[E] 86! then you put the thing in in 1978?

[U] 1978, I'm sorry. Yes it's 1978, and it's an 8 year certificate for $20,000.

Example 4: Violated Binding (hard).

(occasionally [E1], [E2], etc. if more than one expert responds).

Precondition Categories

All four types of **precondition** misconceptions are detectable, but some can only be detected within restricted forms of queries, and others cannot always be distinguished.

Violated preconditions are the simplest ones to understand; if a precondition is supposed to hold at the current time but does not, then a violated precondition would appear to be the obvious explanation. For instance, in Example 5, the user is inquiring about purchasing a certificate of deposit, and asks if the interest rate would be any different if he bought it from a major bank such as Bache and Company. The expert points out that there is a violated precondition with this plan, since the user thinks that Bache and Company is a major bank when in fact it is a brokerage.

[E] On the 2.5 years it [a 30 month certificate] is 14.55% if you buy it from a savings bank or a savings and loan and that compounds to just a shade under 16%.

[U] How about if bought it from a major bank? ... I'm using Bache and Company.

[E] They're not a bank; they're a broker.

Example 5: Violated Precondition.

However, even though violated preconditions are detectable and identifiable regardless of the type of query, other types of precondition misconceptions can occasionally resemble violated preconditions, depending on how the user's statement is phrased. In many cases, **missing** and **substituted preconditions** are indistinguishable from violated preconditions. All three types of misconceptions will be observed as violations of the correct precondition, whether the agent had the correct precondition, whether he was missing the precondition completely, or whether he had a different precondition in its place. This is because in most cases the agent does not explicitly mention any preconditions; the only thing that an outside observer can see is that the correct precondition does not hold. She cannot see what precondition, if any, is included in the agent's version of the plan. She might have some reason to prefer one interpretation over another, but she cannot always decide unambiguously.

The exception to this is when the user does mention a precondition explicitly in his statement. In Example 6, which contains a **substituted precondition**, the first

expert states exactly what she believes the precondition to be, so the listener knows that it is not a simple violation of the correct precondition. The second expert judges it to be substituted (as opposed to an extra precondition which he might have in addition to the correct one) based on its similarity to the correct precondition.

[E1] The subject of "which state" is a really thorny one. Long ago, the Supreme Court ruled you must be a residence [sic] of exactly one state.

[E2] If my memory is correct, you can have the penalties of residency in 2 states simultaneously.

Example 6: Substituted Precondition.

Missing preconditions are a bit trickier; in these cases, the agent must somehow convey his belief that a particular precondition is *not* necessary for a plan, as in the conversation of Example 7. Here, the expert can conclude that the user is missing the precondition of residence/citizenship because he obviously knows that if it *were* a precondition it would be violated, and yet he still believes that his plan may be valid.

[U] I was wondering if anyone had any experience in claiming their parents living abroad as dependent on their tax return. The parents in question are neither US citizens nor permanent residents.

[E] One of the requirements is ... they MUST be resident aliens or US citizens.

Example 7: Missing Precondition.

Observe that in this example, if the user's parents actually were US citizens, and if the user did not mention anything about citizenship in his query, the expert would have no reason to suspect a missing precondition at all. Even though the user would have exactly the same misconception that he currently has, it would be undetectable. This is a general problem with both missing and substituted preconditions: they are only detectable if the precondition does not hold. If the agent has an incorrect precondition or is missing a precondition completely, but does not mention it in his query, and the precondition is coincidentally satisfied anyway, then the misconception will go undetected by the observer.

Unlike substituted and missing preconditions, which usually show up as violated preconditions when they are not mentioned, **extra preconditions** are completely undetectable when they are not part of the user's query. If the agent does not explicitly mention any preconditions, then there is no way of knowing whether he has any extra ones. So the only time the observer needs to worry about extra preconditions at all is when the agent is directly questioning whether a given step has a particular precondition.

For example, the New York State income tax form has a line for claiming a household tax credit. This credit has several preconditions, but quite surprisingly possessing a household is not one of them. It would be very understandable for an agent to have an extra precondition here (that he needs to own or rent a house in order to have a household credit), but the only way that this would come up in conversation is if the agent said something to the effect of "Hey, don't I have to own a house in order to take a household credit?". This can be semantically paraphrased as "Doesn't the step `fill-in-household-credit` have a precondition of `own-house`?".

Consider Example 8, which is taken from the tax transcripts. Note that if the first expert did not specifically mention the incorrect precondition, we would have no reason to suspect that he has a misconception.

[E1] ... Any alien can take all those [standard] deductions and use exactly the same tax laws which apply to citizens and resident aliens, IF he declares that he INTENDS to become a resident alien or citizen!!

[E2] I just received my copy of the IRS tax guide for non-residents.... On page 1 it specifically states: "Intent is not important in determining residency status."

Example 8: Extra Precondition.

Therefore, while no universal simplification can be done to the classification of precondition misconceptions, we can simplify things in the cases where the user does not explicitly mention a precondition. Under these circumstances, we are able to ignore extra preconditions, and we can collapse the remaining three classes into **violated preconditions**. When the user *does* mention a precondition, then all four types of precondition misconceptions are detectable and distinguishable.

Step Categories

Violated steps have already been eliminated from the classification, as explained on page 43.

An analysis of Section 4.2.1 will reveal that **missing steps** have been proven impossible or undetectable in all sections of the plan hierarchy except for two special cases. The first case is with predecessor nodes that are siblings of a substep along the vertical path from the goal to the queried step. We know that these steps must be done before the agent attempts the current step. If one of these has not been done yet, it may be because the agent is missing this particular step from the hierarchy. However, it is not possible to distinguish this case from the one where the agent has this step in the proper place in his hierarchy but has just not completed it for some reason. These instances of missing steps cannot be distinguished from **violated orderings**.

The other special case is with certain queries that implicitly or explicitly deny the inclusion of the queried step in the plan. The agent could do this by listing all of the substeps that he believes are in the plan, or by expressly stating that a particular step is *not* part of the plan. This has already been discussed on page 47, and such cases are both detectable and distinguishable. In Example 9, the user states that there is no alt node in the plan for investing in IRAs that allows for combining deductible and non-deductible money in the same account. Because he specifically mentions this, the expert is able to recognize the missing step and correct it.

[U] My understanding is that it is not permissible to combine deductible and non-deductible IRA money in the same account. Must I disentangle the two accounts, and how do I do it?

[E] My understanding is that it is permitted to do so, but it will result in major headaches as you'll have to retain your IRA records for the rest of your life and engage in non-trivial computations when filing your tax forms.

Example 9: Missing Step.

Extra steps are usually distinguishable from **substituted steps** based on whether or not it is possible to identify the correct step that the substituted step has replaced. In Example 10, the user explicitly mentions each substep in his plan to kill a region in Emacs. The expert tries to match each one to a correct step, and finds that there is

still an extra **set-mark** step that does not match anything.

[U] I have not been very successful in killing any regions yet. I tried 3 times and no luck.

[E] What are you doing?

[U] I set the mark at the beginning, the mark at the end, then esc∧K and doesn't work.

[E] OK the problem is you only set the mark at one end of the region. Do not set the mark twice; it will not work.

Example 10: Extra Step.

In contrast, the user in Example 11 is trying to center the comments in his program by inserting tabs. It is easy for the expert to realize that this incorrect step corresponds to the correct `center-line` step, and that it is not an extra step that the user has *in addition to* the `center-line` step. Of course, it is possible to have a case where the user's step is so different from the correct step that the observer may be unable to find a connection, in which case she may incorrectly diagnose it as an extra step.

[U] What do you want done to the comment headers? Three tabs inserted?

[E] No, they should be perfectly centered. Do that with the "Centerline" command, esc-C.

Example 11: Substituted Step.

Violated optimizations are technically a subclass of substituted steps, but they can be distinguished by the fact that the step itself is not incorrect. In other words, the subplan rooted at the step is valid but suboptimal. In Example 12, the user's plan for purchasing municipal bonds is valid, but it is not the best way to invest his money.

So **substituted steps**, **extra steps**, and **optimization** violations are all detectable and distinguishable categories of step misconceptions, and **missing steps** can be detected and distinguished in certain cases.

[E] Put this money aside in treasury notes.

[U] I thought maybe we should buy municipal bonds with that.

[E] If you buy municipal bonds the interest on your loan will not be deductible. So municipal bonds just don't make sense in this case.

Example 12: Violated Optimization.

Ordering Categories

Section 4.1 stated that **orderings** could be viewed as special types of preconditions. However, unlike preconditions, it appears from a study of the transcripts that people do not tend to make explicit statements about orderings in their queries. Following the analysis of preconditions on page 53, the four types of ordering misconceptions would only be distinguishable if they were explicitly mentioned. So **extra orderings** are undetectable and **violated, substituted**, and **missing orderings** can be collapsed into a single category. As mentioned earlier, certain types of missing steps can also be included in this category.

Example 13 shows a case of a violated ordering from the Emacs transcripts. The user is trying to define a macro, but he forgets to insert the body of the macro before he closes it. Notice how close this example is to a case with a missing step; just changing the wording of the user slightly could give the impression that he didn't think he had to insert the body of the macro at all.

To illustrate a case of **violated mutex**, the user in Example 14 is trying to declare income from profit sharing on his income tax in two different ways: by counting it as a capital gain, and by using a 10-year averaging method. Both methods are acceptable, but only one of them can be used. Since the user is trying to do both, he is violating a mutual exclusion constraint.

Summary

Figure 4.3 summarizes the simplifications that have been made to the classification in this section[3]. The original sixteen categories of plan-based misconceptions were derived directly from the structure of plans presented in Chapter 3. The first part of

[3]The rightmost column of Figure 4.3 will be discussed on page 62.

[E] You can set a macro to change the word. To do this first type ∧X-(. This will start your macro.... Then do a search for the word you are trying to change and make all the appropriate changes for that one particular case as you would normally. When you are done type ∧X-)....

[U] I guess you have to give me the instructions once more. I don't seem to get it right. I type ∧X-(, do a search for that word, type ∧X-), get the binding message, ... and then what?

[E] You don't want to type ∧X-) until you have changed the mistake. Make the changes you want, then type ∧X-).

Example 13: Violated Ordering.

[E] You only get that 10 year advantage [10 year averaging].

[U] Isn't there a portion of that profit sharing which is ordinary income and ...?

[E] Yeah, the amount based upon pre-1976 contributions can be capital gain.... But you can't use the capital gain plus the 10 year advantage.

[U] Oh, I see. You have to use one or the other.

Example 14: Violated Mutex.

PLAN-BASED MISCONCEPTIONS		
Component	**Failure**	**Frequency**
Bindings	Violated	16–21
	Substituted	
	Missing	XXX
	Extra	XXX
Preconditions	Violated	7–14
	Substituted	8
	Missing	16–23
	Extra	4–5
Steps	Optimization	17
	Substituted	11–14
	Missing	5–6
	Extra	12–14
Orderings	Violated	2
	Substituted	
	Missing	
	Extra	XXX

Figure 4.3: A simplified classification of plan-based misconceptions.

Section 4.2 showed that it is only necessary to search for these misconceptions along the vertical path from the agent's goal to his queried step. And the second part showed that the classification can be simplified to ten types of misconceptions that are (at least sometimes) distinguishable from each other and are detectable to an outside observer. These results were derived from directly reasoning about the information available to an outside observer.

4.3 Evaluation of the Classification

Although there is now a theoretical justification for this classification, we still need to demonstrate that this classification accurately accounts for the types of plan-based misconceptions that people actually make. In order to provide this pragmatic justification, this section will analyze the dialogue examples from the transcripts described in Section 1.4 to see how well they fit into the classification.

4.3.1 Transcript Examples of Plan-Based Misconceptions

Even though the purpose of collecting the dialogues in the transcripts was to illustrate examples of plan-based misconceptions, closer inspection revealed that some of the dialogues were not really plan-based misconceptions after all. Of the 111 examples that were originally collected, only 79 conformed exactly to the technical definition given in Section 3.5. There are two reasons why some of the examples were judged not to be plan-based misconceptions: some were not plan-based, and some were not misconceptions.

There are 18 instances where even though the example is plan-based and the user may be experiencing some difficulty, he is *questioning* whether something is possible. If the user questions a fact, then he is not making any claims as to whether it is true or not, so it isn't technically a misconception. For instance, in Example 15, the user seems to be surprised that he can still contribute to an IRA account at age 65. But the way that his question is phrased, he is not actually expressing any misconception. If the wording of the example were changed slightly so that he is definitely expressing a belief that the maximum age limit is lower than 65, then it could be viewed as a substituted precondition.

Many of these examples can be treated exactly as if they are plan-based misconceptions, since their lack of misconception is often just an artifact of the particular way in

[E] Let's at least save 15% of that $4000 that he earned in his job, and put it in an IRA account.

[U] Can you join the IRA when you're 65?

[E] Oh absolutely; up until ... the year in which you're 70 1/2.

Example 15: Queried example; no misconception.

which the dialogues were phrased. They could be converted to legitimate plan-based misconceptions just by changing the wording slightly, and even the original speakers might have rephrased them in that way if they were placed back in the same situation. However, to be safe, these examples were kept separate from the naturally occurring plan-based misconception examples.

There are 14 other examples which do contain actual misconceptions, but which are not plan-based in their current form. These often take the form of isolated statements about the world. In Example 16, the user has an obvious misconception, but there is no mention of a plan. This example could have been classified as a plan-based misconception if additional context had firmly established that the user was referring to a component of a plan, but this context was not present in the actual dialogue.

[U] ... In Pennsylvania, municipalities can not tax nonresidents who work there.

[E] I believe this last statement to be false. The city of Philadelphia has a city wage tax. Every person who works in Philadelphia pays this tax regardless of where they live. In other words if you live in the suburbs, but work in the city, you pay city wage tax!!!

Example 16: Misconception, but no plan.

To judge the effectiveness of the classification, I used the 97 examples that are either plan-based misconceptions or queried examples that could easily be rephrased into the proper form. For each of these dialogues, I located the plan-based misconceptions (some had more than one) and checked to see if they fit into the classification. The results of this analysis are shown in the rightmost column of Figure 4.3. These totals reflect the number of misconceptions (not the number of examples) that fall into each

category.

4.3.2 Ambiguity in Classification

Several of the entries in Figure 4.3 are ranges instead of single numbers. This is because it was possible to place some of the misconceptions 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. In Example 17, the expert is trying to get the user to define a macro, which is not allowed in the current Emacs mode. There are two possible interpretations of this example: either the expert didn't know that macros were disabled by "safe mode 2", or else she didn't realize that the user was in this particular mode. Of course, she herself knows whether this precondition is violated or whether it is missing. But as observers of this conversation, we are unsure. This type of situation was already discussed on page 53.

[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.

Example 17: Violated Precondition or Missing Precondition?

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. In Example 18, the obvious misconception is that the user is trying to invest his money in something that does not exist. But depending on how the hierarchy is constructed, this could appear in two different ways. It may be that the plan hierarchy contains a plan such as

```
(invest-money ?type-of-investment ?amount)
```

and the user is trying to bind the nonexistent object `treasury-twin` to the parameter `?type-of-investment`, which causes a violated binding. On the other hand, it may be that the `invest-money` plan only has a single parameter for the amount of the investment, and that the type of investment is handled by the alt plans (such as

`invest-in-stocks` or `invest-in-treasury-notes`), in which case the user's misconception would show up as a substituted step.

[U] I have a brother that suggested to take out treasury twins.... I never heard it mentioned on this show about treasury twins.

[E] Treasury what?

[U] ... T-w-i-n-s. TWINS, treasury TWINS.... My brother said he has them.

[E] ... I have never heard of TWINS.

Example 18: Violated Binding or Substituted Step?

This example shows that sometimes it is possible to translate one type of misconception into another simply by restructuring the plan hierarchy. This is an artifact of the particular knowledge representation used for plans, and as such is not central to the theory itself. But this translational ability happens to be convenient from an implementation perspective, because it allows us to convert "hard" misconceptions into "easier" misconceptions (see Section 5.8.5).

4.3.3 Unclassified Examples

There are three examples from the transcripts that do not seem to fit into the classification at all. Surprisingly, this is the case even though two of the dialogues are simple variations on an example that *can* be handled.

There were several places in the Emacs transcripts where users experienced difficulty with "setting the mark" (used in tasks such as defining a region and killing a region). Many users thought that there were two marks — one at the beginning of a region of text, and one at the end. But in fact there is only one mark. One dialogue involving difficulty in setting marks was given in Example 10 on page 57, which was used to illustrate an extra step. In both Example 19 and Example 20, the user has exactly the same problem: he thinks that there are two marks that need to be set. But in these examples, there is no extra step that he is referring to. In fact, he never even explicitly refers to the second mark at all, let alone to a step which uses it. He only alludes to the non-existent second mark in a very indirect way, by mentioning "the first mark" (which implies the existence of a second mark) or "only one mark active" (which implies that

there may be other inactive marks). So there is nothing in the dialogues that we can use to pick out a specific failed plan component. And yet they are still plan-based, and they are still misconceptions.

[U] I set the first mark at the bottom of what I wanted, so I didn't want to delete anything below that. I tried typing esc g a few lines below but saw no response.

[E] Hmmm. Let's do that again. Set the mark (there is only one mark. THE mark), move over one word or line, and type esc-g. The point (cursor) should move back and forth.

Example 19: A difficult example of setting two marks.

[U] I want to move a block of text as a unit. How do I do it?

[E] Mark out the region (do be sure to check that you've marked the right region with esc-G), kill it, move to where you want it, and unkill it (∧N). Comprende?

[U] Is only one mark active at a time?

[E] No. That's not quite the idea. There is only one mark. You kill a region BETWEEN the point & mark.

Example 20: Another difficult example of setting two marks.

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 might be possible to find a complex combination of missing steps, violated bindings, substituted steps, and missing preconditions that explains each expert's mistake. 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.

4.3.4 Conclusions

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 we will see later that the "structurally ambiguous" examples are actually desirable for the process of automating the detection of these misconceptions.

Assuming that the examples in the transcripts are representative of the sorts of mistakes that people make with their plans in real life, this demonstrates that the classification developed here is a reasonable way to organize plan-based misconceptions.

The main advantage of this type of classification over the "belief-based" classifications is that misconceptions are associated directly with a specific component of a plan, rather than being treated as more isolated incorrect beliefs. This will make identifying plan-based misconceptions much more straightforward, since the detection process can now be completely combined with the plan recognition process described in Chapter 5.

4.4 Comparison to Other Classifications

Pollack [83] suggests three classes of plan-based misconceptions. **Unexecutable 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 Example 21; 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 Example 21 as a particularly bad extra step. Clearly the user believes that standing on his head is part of a phoning plan, and since

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

Example 21: Incoherent query.

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. It is more appropriate to handle this with probabilities, as described in Chapter 6.

Quilici [85] 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 3.4.1 only represents effects at the atomic level (via the add/delete lists), my classification can only address special cases of violated effects.

Van Beek [110] uses a classification similar to that presented by Joshi [48]. 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

	1	2	3	4	5	6	7	8	9	10	11	12
Calistri	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	n	s
Pollack	y			n	s				n	n	Y	n
Quilici	y				y			n	n	n	n	Y
vanBeek	y			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)

Figure 4.4: Comparison of the four classifications.

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 4.4 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". A lowercase "y" means that several of these classes have been collapsed into a single class. A lowercase "s" means that the classification recognizes only special cases of this type of misconception.

This figure illustrates that my classification encompasses a superset of the other three systems. The one type of misconception that it does not represent is "incoherent plan", which is indirectly addressed by probability measures on extra steps. It is quite significant that I handle categories that are not addressed at all by the other three classifications. It is less significant that several of my 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.

Chapter 5

Algorithms for Robust Plan Recognition

Abstract: *The traditional plan recognition problem of Chapter 3 can be solved in a straightforward manner by a graph-searching algorithm such as* **A***. *A few simple extensions allow the algorithm to handle situations in which the user has an incorrect plan. However, it turns out that recognizing structural errors is computationally much more complex than recognizing plans with simple constraint violations. This chapter presents three general algorithms, starting with a direct adaptation of* **A*** *for recognizing correct plans, and progressing to a more robust algorithm that also recognizes invalid plans*[1].

5.1 Plan Recognition as Graph Searching

The plan recognition problem can be viewed as a particular application of the general graph searching problem. The plan terminology can be mapped directly onto graph terminology in the following way. As Figure 1.1 illustrated, the hierarchy of plans is structured as a directed *AND/OR* graph, where plans are composed of subplans (*AND* nodes) and where there can be alternate ways to execute a plan (*OR* nodes). The input to the problem is this graph (the hierarchy), along with a set of marked nodes (steps that the user has mentioned). The task is to search the graph for a path (explanation)

[1]Much of the material in this chapter has been published in [19].

that extends from a root of the graph (the agent's top-level goal) to a specific node (the observed action), passing through all of the marked intermediate nodes.

Graph searching is quite common in AI, but it usually takes a slightly different form. Often, the graph that is searched does not consist of plan steps, but rather consists of problem states. The graph is traversed by applying an operator that transforms one state of the world into another. The object is to find a sequence of operators that transforms the initial (current) state into the desired (goal) state. This formulation of the search problem is used in classical problems such as the 8-puzzle and the "blocks world" [75, Ch. 2–3], as well as much of the work in robot path planning [58, 59, 107], and with slight adaptations can also be used in game-playing situations such as chess [78]. However, if we abstract away from the problem just slightly, we notice that the same nodes that are interpreted as state descriptions in a state space can also be interpreted as plans in a plan hierarchy [26, Ch. 5]. The graph properties are exactly the same, so it should be possible to apply the same graph methods to both problems.

The **A*** algorithm can be applied directly to the "Perfect Plan Recognition" problem to find an optimal explanation for the agent's query. But there are two modifications that need to made in order to handle plan-based misconceptions in the "Robust Plan Recognition" problem. Pathfinder is a particular implementation of this algorithm which has been optimized to deal with misconceptions in a more efficient way.

5.2 PR-1: A* for Perfect Plan Recognition

The Perfect Plan Recognition problem, as discussed in Section 3.2, can be stated as follows:

> *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 do not contain any misconceptions.*

The **PR-1** algorithm presented here is a direct adaptation of the **A*** algorithm [45, 101], as described by Pearl [81, p. 64 & Ch. 3.1], applied to the Perfect Plan Recognition problem. One change that is necessary in order to adapt the normal version of **A*** for perfect plan recognition is the input to the problem. While the original **A***

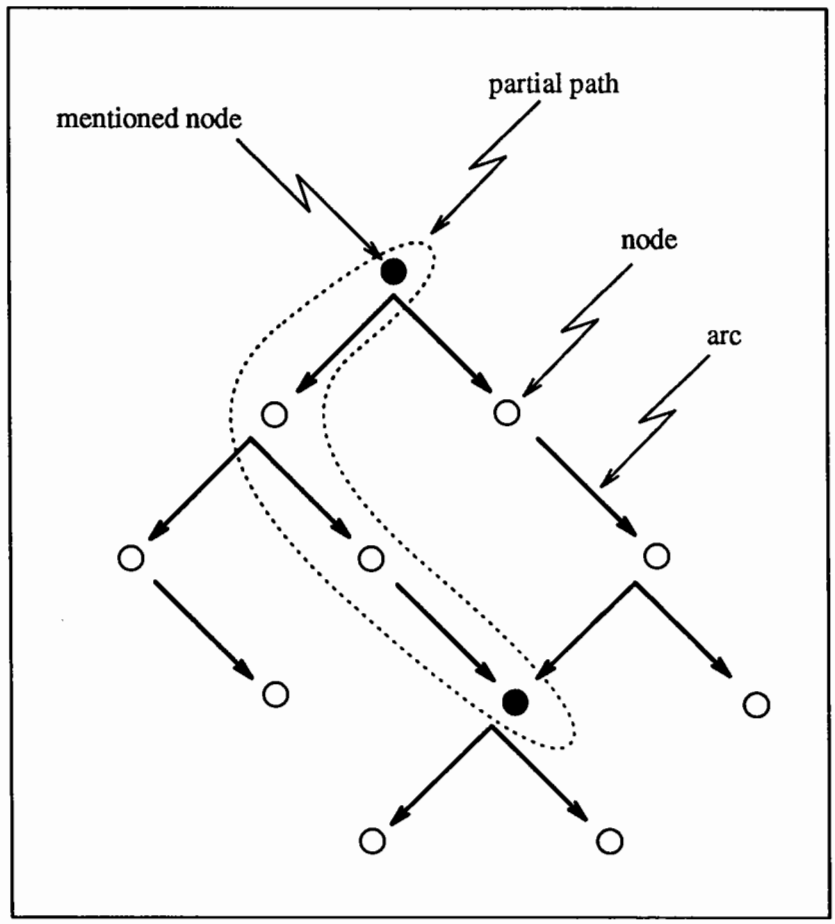

Figure 5.1: A partial path in the plan hierarchy.

only had the start and destination nodes (the top-level goal and the actual queried step) as input, we also have other known points along the path (the other steps that the agent has mentioned). Since we know that the solution path must pass through these points, we can change the basic search unit from a single node to a partial path between two consecutive mentioned nodes, as shown in Figure 5.1.

The **A*** algorithm 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 start node, 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 that no constraints have been violated. For example, a precondition for one of the steps may not hold in the current state of the world. If there are any violations, we discard this path, since it could not be part of a correct plan (remember, we are currently doing perfect plan recognition). Otherwise, if this partial path has been verified successfully, we compute a heuristic function f. This function estimates our confidence that this partial path will be part of the final answer. 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 a destination node, in which case we have successfully recognized the agent's plan, or until we run out of partial paths to expand, in which case we have failed.

This algorithm is shown in Figure 5.2. There are several things to note here. First of all, the algorithm is purposely ambiguous about the **Expand** procedure. The successors of p can be generated in any way that is appropriate for the particular domain. Some domains are structured in such a way that a special-purpose method can be used to generate these successors; it may even be possible to precompute the necessary partial paths. In other situations, the most reasonable approach might be to call **A*** recursively with the next mentioned step as a destination.

Another deliberate ambiguity is in the choice of "start" and "destination"; it is equally possible to proceed top-down, from the goal to the mentioned step, or bottom-up, from the mentioned step to the goal. The choice of direction may be influenced by the relative size of the forward and backward branching factor of the plan hierarchy. It is most efficient to search in the direction with the smallest branching factor, since each branch represents a point where the search may need to split. If the domain contains very complex subplans, this signifies a high forward branching factor, and a bottom-up approach might be more efficient. If there are certain very common actions that are found in almost all plans, this signals a high backward branching factor, in which case a top-down approach would make more sense.

1. OPEN $\leftarrow$ start
   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 a destination node, exit with the complete path formed by tracing the pointers back to **start**.
5. Otherwise, **Expand**(p):
   - **5.1.** Generate all successors of p (all partial paths from endpoint of p to the next known node), with pointers back to p.
   - **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, remove it.
     - **5.2.3.** Otherwise, estimate $f(p')$ and add p' to OPEN.
6. Go to step 2.

Figure 5.2: The **PR-1** algorithm.

Finally, the choice of a partial path with "optimal" $f(p)$ does not distinguish whether the heuristic function f should be maximized or minimized. In fact, the algorithm doesn't specify how to calculate $f(p')$ for the successors at all. The traditional **A*** algorithm defines a minimizing function $f(p') = g(p') + h(n')$. The function $g(p')$ is the earlier cost of expanding the path from the start node to p, plus the new cost of extending p to p'. The function $h(n')$ is an estimate of the cost of expanding p' to a destination node. However, these requirements are not necessary to the algorithm [10, 11, 33]. It is possible to define f to be either a minimizing function, where f measures the cost of a path, or a maximizing function, where f measures the merit of a path. It is also possible for g and h to be multiplicative instead of additive. In fact, there are only two strict requirements on the heuristic function. In order to guarantee an optimal answer, $f(p')$ must be *optimistic* and *monotonic*. This means that for a maximizing heuristic, f must always overestimate the merit and must be non-increasing. And for a minimizing heuristic, f must always underestimate the cost and must be non-decreasing. Chapter 6 will describe the details of a particular heuristic function based on probabilities.

5.3 Alternatives to A*

There are different search techniques, depending on the type of graph and the type of desired solution. Normally, if the graph is treated as an *AND/OR* graph, **A*** is not appropriate and it is necessary to use a different version known as **AO***. But even though the plan hierarchy is structured as an *AND/OR* graph, it is still appropriate to use **A*** instead of **AO***, because the searching process does not make any distinction between the two types of nodes. During the construction of the solution path, it does not make any difference whether a particular node is a substep or an alt; all that matters is that the node lies on the path. So the plan hierarchy can be treated as a normal *OR* graph, and the **A*** method is appropriate.

There are two popular alternatives to **A*** that are frequently used for searching. One, iterative-deepening **A*** (**IDA***), employs a space-time tradeoff to save memory at the expense of reconstructing all nodes along a partial path at every iteration [57]. This is only effective when node evaluation is very cheap, which is not the case in plan recognition. Sen [98] proposes a new method which combines the properties of **A*** and **IDA***; it is similar to **A*** unless there are memory restrictions, in which case it runs in

a hybrid mode.

The other alternative is to relax the optimality requirements, and search for a *satisficing* solution instead of an optimal solution [32, 33, 78]. This can produce a much more efficient algorithm, but it is not appropriate for plan recognition. There may be many satisficing plans that provide a consistent explanation for the user's query but are extremely unlikely. We need to be able to distinguish the most likely explanations from the unreasonable explanations, so the ability of **A*** to give an optimal solution is very important and cannot be abandoned.

5.4 Misconceptions and the Plan Recognition Process

Chapter 4 developed a classification of ten categories of plan-based misconceptions. An algorithm for robust plan recognition must be able to handle all ten of these cases. Based on how they affect the algorithm, it is convenient to regroup these misconception types into two broad classifications: **structural** misconceptions and **constraint** misconceptions. The structural misconceptions are substituted steps, missing steps, and extra steps. Plans that contain structural misconceptions will be referred to as **ill-formed**. The remaining seven misconception types (violated bindings, the four precondition misconceptions, violated orderings, and violated optimization) are called constraint misconceptions because they all involve constraints that are placed on the plan. Plans that contain constraint misconceptions will be referred to as **violated**, since this group encompasses all of the violated misconceptions. Plans which have no misconceptions at all are called **valid**.

The crucial difference between structural and constraint misconceptions is how 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.

To illustrate this difference, let us once again consider the graduate student requesting a W2 form for his fellowship. In the case where the student has a general fellowship, the correct plan for recording his information does involve performing a

`get-obj` step, but he should be getting a 1099 form instead of a W2 form (an example of a violated binding). This path can still be found in the correct plan hierarchy, because there is an arc connecting the `fill-in-general-fellowship` node and the `get-obj` node, but this particular plan is violated because of the violated binding. On the other hand, if the student has a Brown fellowship, there is no `get-obj` step at all under `fill-in-brown-fellowship`. If he really does believe that he is supposed to get something for his Brown fellowship, this constitutes an extra step, and no amount of searching in the correct hierarchy will find such a path. We would need to add an extra arc in order to recognize this plan, which changes the structure of the hierarchy and makes this an ill-formed plan.

There are two separate modifications to **PR-1** that need to be made in order to handle the two general classes of misconceptions. **PR-2** handles constraint misconceptions, and **PR-3** extends this to handle structural cases as well.

5.5 PR-2: Allowing Violated Plans

PR-1 can be modified to deal with violated plans fairly easily by changing the **Expand** procedure (Step 5). The key is that plans which only contain constraint misconceptions will already be generated by **PR-1**, and will even be recognized as invalid in Step 5.2. This verification process actually tests for the presence of all seven types of constraint misconceptions. Bindings are checked to make sure that they are consistent with any pre-stored values. Preconditions are verified to ensure that they hold in the current state of the world. If the user mentioned any preconditions in his query, these are checked to make sure that they actually are preconditions for the current plan. And temporal constraints are checked as well.

The difficulty with **PR-1** is that when a violation is encountered, the improper plan is thrown out in Step 5.2.2. Once agents are allowed to have faulty plans, we can no longer discard a path just because it has a violation, because it could still be the path that the user is intending to take. So rather than throwing out a path when a constraint misconception has been detected, we will instead add the successor path p' to the `OPEN` list whether or not it is part of a correct plan. In order to keep track of the fact that we have discovered a misconception in the agent's plan, we will add appropriate tags to the path which document the misconceptions. These misconceptions are also taken into account when calculating the heuristic function f. This gives us the ability to

1. OPEN ← start
   CLOSED ← ∅.
2. If OPEN = ∅ 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 a destination node, exit with the complete path formed by tracing the pointers back to start.
5. Otherwise, **Expand**(p):
   - **5.1.** Generate all successors of p (all partial paths from endpoint of p to the next known node), with pointers back to p.
   - **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 appropriate documentation to the path.
     - **5.2.3.** Regardless of violations, estimate $f(p')$ and add p' to OPEN.
6. Go to step 2.

Figure 5.3: The **PR-2** algorithm.

prefer plans with fewer (or less serious) misconceptions in the proper circumstances.

This modification produces the **PR-2** algorithm, as shown in Figure 5.3. Changes from the **PR-1** algorithm are underlined.

5.6 PR-3: Allowing Ill-Formed Plans

Ill-formed plans are a bit trickier to handle. While violated plans were found by **PR-1** and then discarded, plans with structural misconceptions were not even generated by **PR-1** in the first place. Since ill-formed plans do not exist in the correct plan hierarchy, **PR-1** will never search for them. So the method for generating successors in Step 5.1 must be modified to include additional partial paths that do not 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. But in order to keep the algorithm as general as possible, the

1. OPEN ← start
   CLOSED ← ∅.
2. If OPEN = ∅ 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 a destination node, exit with the complete path formed by tracing the pointers back to **start**.
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 error nodes 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 appropriate documentation to the path.
     - **5.2.3.** Regardless of violations, estimate $f(p')$ and add p' to OPEN.
6. Go to step 2.

Figure 5.4: The **PR-3** algorithm.

particular method of generating the ill-formed paths is left up to the implementation.

This results in the **PR-3** algorithm in Figure 5.4. Constraint misconceptions are handled exactly as in **PR-2**. New changes for dealing with structural misconceptions are underlined. **PR-3** is now able to handle perfect plan recognition as well as all ten types of plan-based misconceptions discussed in Chapter 4. Therefore, **PR-3** qualifies as a "robust" plan recognition algorithm, since it can handle plan-based misconceptions even if they are new and unanticipated.

5.7 Analysis

It is instructive to analyze each of the three plan recognition algorithms in terms of the following properties:

- **Admissibility**: is the algorithm guaranteed to find the best solution? This can actually be three separate questions, based on whether the user's plan is valid,

violated, or ill-formed. For example, an algorithm could be admissible for valid plans but not for violated plans.

- **Complexity**: how long does the algorithm take to find the answer in the worst case? Alternately, how many nodes of the graph does the algorithm expand before finding the correct path?

Note that having an admissible algorithm is *not* the same as having an admissible heuristic function. The only requirement in the latter case is that the heuristic is optimistic and monotonic; it does not have any requirements for applying this heuristic to all appropriate paths. An admissible algorithm must also guarantee that all of the relevant paths will be considered. We will assume that all algorithms always use an admissible heuristic function; the probabilistic heuristics that are actually used in the implementation of the Pathfinder program are admissible.

The A* algorithm

Pearl has shown that if the heuristic function f is admissible, then **A*** is guaranteed to find an optimal solution path if one exists [81, p. 86]. This admissibility result only holds for *valid* plans, since **A*** does not even have a concept of invalid plans.

He has also shown that "a necessary and sufficient condition for maintaining polynomial search complexity is that **A*** be guided by heuristics with logarithmic precision" [81, p. 184]. The complexity here is measured by the number of expanded nodes relative to N, the length of the shortest path acceptable for the answer. For simplicity, assume that this path will be roughly as long as the average depth of the graph (i.e., for the plan hierarchy, assume that the top-level goal is a generic plan and that the mentioned step is an atomic action). This means that **A*** will be polynomial in the depth of the plan hierarchy if and only if $f(p)$ is logarithmically precise. In other words, if $\phi(f)$ describes the growth of the typical error in $f(p)$, then $\phi(N)$ must grow no faster than $c(\log N)^k$ for arbitrary constant k to maintain polynomial complexity. This is a very stringent demand on the precision of the estimation, and it requires choosing the heuristic function carefully. But it still allows **A*** to be a polynomial algorithm under proper circumstances.

As long as f is an admissible heuristic, it is possible to implement **A*** in such a way that it will never visit a node more than once, irrespective of the precision of the

heuristic. So for an upper bound, **A*** is guaranteed to be no worse than exponential in the depth of the graph.

The PR-1 algorithm

PR-1 is a direct adaptation of **A***; it generates and selects its nodes in the same manner as **A***, and it has the same requirement that the heuristic function is admissible. So **PR-1** is also admissible for correct plans.

The number of nodes expanded by **PR-1** may actually be *less* than **A***, since **A*** does not have any explicit verification process. Although **PR-1** and **A*** both generate and select nodes in the same way, **PR-1** has the additional step of pruning violated nodes, which can reduce the final number of expanded nodes.

The PR-2 algorithm

The results for **PR-2** are also encouraging. Recall that the change to produce **PR-2** was that paths were not pruned even if they contained violations. The heuristic function takes into account any violations that are detected, and as long as f is still required to be overly optimistic, the answer is still guaranteed to be optimal. This admissibility result now holds for both correct and violated plans. And since the original **A*** didn't prune violated paths in the first place (it didn't even have any concept of violation), we are not expanding any more paths than **A***. Assuming that constraints can be checked in an efficient manner, this means that **PR-2** is no worse than **A*** in complexity.

The PR-3 algorithm

Unfortunately, the results for **PR-3** are not as favorable. To produce this algorithm, the successor function was modified to generate additional partial paths. These paths do not even exist in the original plan hierarchy, so there is no way that **A*** would have expanded them. But since the algorithm does not specify how these ill-formed paths are generated, the **Expand** procedure could conceivably produce an exponential (or even larger) number of additional paths at every iteration of the algorithm. And even if the heuristic function is extremely precise, so that we will only expand a node if it lies along the solution path, simply generating the ill-formed paths in the first place may violate the polynomial runtime.

	PR-1	PR-2	PR-3
Admissible:			
for valid plans	1	1	1
for violated plans	No	1	1
for ill-formed plans	No	No	3
Complexity	2	2	3

1: Admissible iff $f(p)$ is optimistic.
2: Polynomial iff $f(p)$ is logarithmically precise.
3: Depends on both $f(p)$ and the successor function; possibly not admissible, probably at least exponential.

Figure 5.5: Summary of the analysis.

We now have problems with the admissibility claim as well. As long as the heuristic function is optimistic, we will still find the optimal path *among those generated by the successor function.* So **PR-3** is still admissible for valid and violated plans. But Step 5.1, which generates the successors of a partial path, does not guarantee that all of the relevant ill-formed paths are generated. The true optimal path may have involved an ill-formed partial path that was never generated because the generation method itself was incomplete. In fact, it may even be the case that a non-exhaustive admissible successor function does not exist at all.

So the properties of **PR-3** are strongly tied to the method by which the successor function generates ill-formed partial paths. There appears to be a fundamental tradeoff between efficiency and admissibility. If too many partial paths are generated, we lose all hope of achieving a reasonable runtime. On the other hand, if we start restricting the successor function, we run the risk of missing the optimal solution. In order to handle ill-formed plans in a sensible manner, the successor function needs to be designed in such a way as to minimize both the number of extraneous paths generated and the chances of missing the actual path that the user chose.

Summary

The results of this analysis are summarized in Figure 5.5. This shows that the **A*** algorithm can be directly applied to the basic plan recognition problem while keeping its desirable properties of admissibility and the possibility of polynomial runtime. Two

relatively minor modifications can extend the robustness by allowing the algorithm to recognize situations in which the user has an incorrect plan. Violated plans are those plans that are formed correctly but are inappropriate in the current situation because of a mistake involving preconditions, bindings, temporal constraints, or optimality. These are easy to handle, and an algorithm to identify them can still retain the desirable properties of the **A*** algorithm. Ill-formed plans are those plans whose structure is incorrectly formed, because of a substituted, missing, or extra step. There is a fundamental difference between violated and ill-formed plans; identifying ill-formed plans is inherently much more difficult, and we cannot guarantee admissibility of the algorithm for this class of plans.

Two critical decisions in implementing the algorithm are the selection of a powerful heuristic function and the selection of an appropriate successor function. The first function is the means by which the search space is pruned, and the second function is the means by which the original search space is expanded to include ill-formed paths. An optimal heuristic function would guarantee that the minimum number of partial paths will be selected for expansion, and an optimal generation function would guarantee that the partial paths that *are* selected will generate a minimum number of successors. If either function is too restrictive, the algorithm will no longer be admissible. If they are too permissive, the algorithm becomes intractable.

5.8 The Pathfinder Program

The general algorithm for robust plan recognition presented in **PR-3** has been implemented in a program called Pathfinder. Pathfinder is designed to be a component of an integrated natural language system, as described in Section 3.3. Its purpose is to act as an interface between the semantic interpreter and an application program such as a command executor or expert system. Rather than just blindly executing the user's commands or retrieving the information that the user requested, Pathfinder analyzes the user's goals and his command or query and attempts to determine if the request the user is making is reasonable.

There were a number of design decisions that were purposely left ambiguous in the presentation of **PR-1**, which need to be resolved in the Pathfinder program. One of these ambiguities was the selection of the "start" and "destination" nodes. Usually, this issue is decided based on the branching properties of the hierarchy. In the sample

plan hierarchy that has been developed for Pathfinder, the average forward branching factor was slightly higher than the backward branching factor. But there were a few very common nodes that had backward branching factors that were twice as high as the largest forward branching factor. So Pathfinder currently runs in a top-down fashion; it starts with the user's top-level goal and works toward a destination of the mentioned step.

Another ambiguity in **PR-1** was the method by which successors were generated. In Pathfinder, well-formed successors are pre-computed and stored in a hash table for efficiency purposes. Each node in the hierarchy has an entry in the hash table which stores all possible partial paths from the root (generic plan) to the node. This information can be used to extract paths between arbitrary nodes very quickly[2]. Ill-formed successors (partial paths involving substituted or extra steps) are generated separately, and will be described later.

Finally, the heuristics that are used to guide the search are implemented using probabilities. For each path that is generated, Pathfinder calculates the probability that the user would actually be following the corresponding plan, and also calculates the probability that the user would actually have the misconceptions that the path entails. This provides an accurate method of choosing among competing explanations for ambiguous plans, as well as a means of measuring our confidence that we have successfully recognized the user's plan. The use of probabilities in Pathfinder will be discussed in detail in Chapter 6.

5.8.1 The Basic Algorithm

The algorithm used in Pathfinder is basically an optimized version of **PR-3**. While **PR-3** can deal properly with incorrect plans, it does not always verify or expand partial paths in an intelligent manner. It turns out that it is much more efficient to split the **Expand** and **Verify** tasks into two separate procedures. In **PR-1**, it made sense to verify partial paths as soon as they were generated, because violated paths were immediately pruned off, reducing the search space. But now paths are kept whether or not they contain any violations. Since the verification process may take up a significant proportion of the time spent on each node, it would now make more sense to postpone verification, and only do it on the partial paths that are likely to be included in the

[2]Pathfinder is also capable of storing these paths directly, rather than deriving them from the paths to the root.

final answer.

In order to separate these two processes, Pathfinder maintains a third list, called the GENERATED list. Partial paths are placed on this list if they have been generated but have not yet been verified. The evolution of a partial path progresses from GENERATED (after it is created) to OPEN (after it is verified) to CLOSED (after it is expanded).

At any iteration of the algorithm, the most likely partial path can be selected from either OPEN *or* GENERATED. If the current partial path is from OPEN, it is removed and placed on CLOSED as before, but during the **Expand** procedure its generated successors are placed on the GENERATED list without being verified. If the current partial path is from GENERATED, it is verified and placed on the OPEN list without being expanded. The Pathfinder algorithm is shown in Figure 5.6. As before, the changes from **PR-3** are underlined. Although Step 2 (the failure condition) is still included, it will never be used in the current implementation. Because of the way that the successor function works, Pathfinder is always guaranteed to find *some* path to the queried step (in the worst case, just a sequence of extra steps from the goal to the queried step).

One consequence of this new algorithm is that we have to estimate $f(p)$ before p has been verified (in order to choose the most likely unverified partial path from GENERATED). Once p is selected to be verified, $f(p)$ is recalculated to reflect any misconceptions that have been found along the partial path. This involves evaluating $f(p)$ twice for any path that comes from the GENERATED list. So two properties have to hold in order for this method to be feasible: the process of evaluating $f(p)$ (at least the first estimation of $f(p)$) should be inexpensive relative to verifying p itself, and the first estimation should still be accurate. The first property guarantees that the overhead of recalculation is justified for individual partial paths, and the second guarantees that it is justified in the long run by reducing the number of paths that are verified. Both of these properties hold for the probabilistic interpretation that is described in Chapter 6.

For the following discussion, assume that the user has given a "normal" query; i.e. that he is not mentioning any preconditions or missing steps. These special types of queries will be handled later.

5.8.2 Finding Constraint Misconceptions

Pathfinder generates partial paths between each pair of consecutive mentioned steps. These partial paths are retrieved from the hash table associated with the plan hierarchy.

1. GENERATED $\leftarrow \emptyset$
   OPEN $\leftarrow$ goal
   CLOSED $\leftarrow \emptyset$.

2. If (OPEN $= \emptyset$) and (GENERATED $= \emptyset$) then exit with failure.

3. Remove partial path p with optimal $f(p)$ from either OPEN or GENERATED; if p was from OPEN, place on CLOSED.

4. If p was from OPEN and the endpoint of p is the queried step, exit with the complete path formed by tracing the pointers back to **goal**.

5. Otherwise, if p was from OPEN, **Expand** (p):

   5.1. Generate all successors of p (all partial paths from the endpoint of p to the next mentioned node, plus other ill-formed partial paths), with pointers back to p. If a successor is ill-formed, add appropriate documentation to the path.

   5.2. For each successor p' of p, estimate $f(p')$ and add p' to GENERATED.

6. Otherwise (p was from GENERATED), **Verify**(p):

   6.1. Check p for violations.

   6.2. If p contains any violations, add appropriate documentation to the path.

   6.3. Regardless of violations, recalculate $f(p)$ and add p to OPEN.

7. Go to step 2.

Figure 5.6: The Pathfinder algorithm.

Bindings are then propagated through the partial path from the mentioned step so that all steps receive the values that the user mentioned. As each partial path is constructed, its nodes are checked for constraint misconceptions.

Violated Preconditions

First, the preconditions of each step are verified. This is done by proving that the propositions specified by the preconditions hold in the current state of the world. If preconditions are allowed to be arbitrarily complex, we would need the power (and expense) of a full theorem prover to verify them. So to permit efficient verification, Pathfinder only allows preconditions to be constructed in a few ways. A precondition can be a simple proposition about the state of the world, which is checked by comparing the proposition to a database containing all relevant facts about the world. A precondition can also be an equality statement or an *isa* statement about two objects. And preconditions can be combined by using conjunction, disjunction, and negation.

Violated Orderings

In addition to checking the preconditions, temporal constraints are also verified. This involves checking the partial orderings to make sure that all steps that must precede the current step have actually been completed, and that no mutual exclusion constraints have been violated. Since Pathfinder is building this path top-down, the orderings of the ancestors have already been verified. The orderings within the subtrees of these ancestor nodes don't need to be checked because if there hasn't been a violated ordering at the current level, then the step must have already been done correctly, so there couldn't be any violated orderings in the subtree. Therefore, the only orderings that need to be checked are the substeps of the parent of the current node. If the parent is an alt node instead of a substep node, then Pathfinder checks the alt children of the parent to verify that no mutual exclusion violations have occurred.

This is a relatively small number of nodes to check. They can be checked very efficiently if we assume the presence of a table that lists all steps that have been completed, along with their times of completion. There are certain problems with building such a table: we can't simply enter all observed or mentioned steps, since these may be at various levels of abstraction, and certain combinations of steps may complete higher-level plans. But it turns out that it is possible to combine the construction of

this table with the plan recognition process with very little overhead. When Pathfinder is called with a step that is known to have taken place (as opposed to a query about a future step), it will mark that step as having occurred. During the reconstruction of the path through the hierarchy, the ancestors can be checked to see if this step has completed the higher-level plan. This will construct the full table of all completed steps.

Violated Bindings

Parameters are also checked during this phase 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. This is because the mentioned steps are the only points where the user is providing input; the bindings at all other steps will be specified by the step's parent, and so must be derived either from a pre-stored (correct) value or from a failure at an upper-level mentioned step which would have been detected earlier. Therefore, since mentioned steps only occur at the endpoints of the partial paths, it is only necessary to check this endpoint. 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.

Violated Optimization

Pathfinder does not currently check for optimization violations. This is because so far no one has been able to find any way of doing it efficiently. By definition, finding optimal or nearly optimal solutions to a problem requires planning capabilities. Since this thesis is about plan recognition rather than planning, we will not worry about the problem of judging whether the user's plan is optimal.

A "complete" solution to this problem would involve embedding a full planning module within the plan recognition system, which would generate all ways of achieving the current goal. Then each of these completed plans would be run through a full simulation system, which would simulate the execution of each plan in the current state of the world. It would assign some measure of optimality to each of these results, then simulate the user's plan and compare that optimality measure to the best solution that the planner generated. This is extremely expensive and complex, and it is certainly

not something that belongs in a plan recognition module.

One possible alternative is to attach some sort of "canned" utility score to each branch of an alt node, where the optimal path is the one with the highest utility. But these scores appear to be highly context-dependent, which would preclude pre-storing them directly in the plan hierarchy, and it may be quite difficult to generate them automatically at runtime. This is still an area of future research.

5.8.3 Finding Ill-Formed Misconceptions

There is one situation where it is easy to realize that the user has an ill-formed plan. This situation is signalled by the fact that no path can be found which connects two mentioned steps in the user's input. This means that the user claims there is a path between two steps, but the correct 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.

Deciding whether the mentioned step should be replaced with something else or is completely extraneous, 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[3], it is fairly simple to handle each case individually.

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

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

(where a is the top-level goal and d is the queried step), and assume that Pathfinder has already expanded the path from a down to b. It will then continue by trying to expand the path from b down to c. If Pathfinder cannot find any extension at all, it means that there is something wrong with either b or c. There are four possibilities: either b is an extra or substituted step, or c is an extra or substituted step.

The first case is actually not possible after all; since we have already found a path from a to b, we already know that b is a legitimate part of the plan for a, so it cannot be an extra step. But it is possible that the user could have had c as an extra step under b. So assume that Pathfinder has identified c as an extra step. Then an appropriate error message will be inserted, and it will continue ahead, trying to expand the path from c down to d. The reason it continues expanding the path is that it may be able to

[3]See page 92.

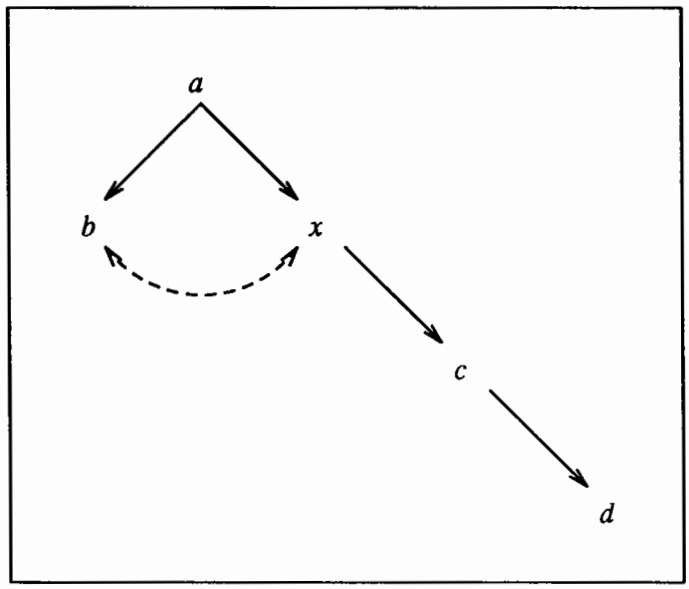

Figure 5.7: Pathfinder considers a possible substituted step.

find additional misconceptions in the segment from c to d, which can then be pointed out to the user.

Alternately, assume that b has been identified as a substituted step (the case with c as a substituted step is handled analogously). In this case, rather than trying to connect b to c, Pathfinder will ignore b (since it is an incorrect step), and will instead try to find a different path to c. It will then throw away the original path from a to b, and will insert an error message stating that instead of doing step b, the user should have done the corresponding step in the new path. For example, the original hierarchy might be structured as in Figure 5.7. There is no way to get from b to c, but there is a way to get to c by passing through x, and the user might have been confused between b and x.

The selection of the appropriate new path is obviously important. Currently, Pathfinder considers all "reasonable" substitutions for b, with "reasonable" being defined by the heuristic function. It then retains only those paths that still pass through a and c, since the user mentioned those steps and they must still be part of the path. This method is not guaranteed to be admissible, since it might never consider the user's actual path. For instance, the user's path might actually substitute b for another step y, which then has c as an extra step. But Pathfinder will only miss the user's path when the user has at least two incorrect mentioned steps in a row, in which case his

plan is so seriously flawed that there is very little hope of correctly deciphering his intentions anyway.

5.8.4 Multiple Possible Paths

This method of handling substituted steps is a bit too simplistic. There are occasions where a substituted step should be proposed even when a path already exists in the hierarchy.

Recall the original discussion of the graduate student example in Chapter 1. There was one possible interpretation where the student was simply confused between W2 forms and 1099 forms. This corresponds to the path through

`(do-taxes)` $\rightarrow \cdots \rightarrow$ `(fill-in-fellowship f235)` $\rightarrow \cdots \rightarrow$ `(get-obj form-w2)`

with a violated binding on the last step. But there was also another explanation where the student understood about the forms but was confused between fellowships and wages. So it is also possible to construct a *different* path:

`(do-taxes)` $\rightarrow \cdots \rightarrow$ `(fill-in-wages f235)` $\rightarrow \cdots \rightarrow$ `(get-obj form-w2)`.

This path avoids the earlier violated binding, but now contains a substituted step (substituting `fill-in-wages` for `fill-in-fellowship`) and a violated precondition (`f235` is not a type of wages).

But if Pathfinder only proposes substituted steps when there is no connecting path in the hierarchy, then it would never find the second explanation. Since the user is capable of having a substituted step anywhere in his plan, even if a separate connecting path exists using his mentioned step, this argues that we should search for possible substituted steps at every possible location. In fact, Pathfinder currently does something very similar to this. But unfortunately, this could get very expensive. So Pathfinder can be configured to take a compromise position: it would only look for possible substituted steps when a misconception has already been 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 example also illustrates the potential danger of ambiguous plans. There is more than one possible explanation for the user's mistake. But only one of them is the *real* explanation, and it is very important to choose the correct one. 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 might think that *all* wages are supposed to get 1099 forms.

This can be taken care of by using the probabilities in Chapter 6 to select the most likely explanation. But this also raises a new concern. The best-first search method that Pathfinder uses must be guided by a monotonic heuristic function, and it is not obvious that Pathfinder can provide such a function the way things are currently formulated. 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 which lowered the probability, but since these steps are no longer part of the path, the probability should no longer be influenced by the misconceptions. So the total probability for the path may *increase* after a substituted step has been found, which violates the monotonicity requirement.

It turns out that the 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

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

Rather than starting off by finding a connection between a and b, Pathfinder will start off with *two* options:

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

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 find a connection from a to b in the hierarchy. If it succeeds, it will produce an expanded path with an associated score. If it fails, it will produce a

degenerate path with b 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 b and c,
- assume that c is an incorrect step and try to connect b to d, ignoring c, or
- assume that b is an incorrect step and try to connect a 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. Whenever the old version would have backtracked and skipped a step, the new version already has the skipped-step alternative present as one of the "red herring" paths. It is true that in order to avoid backtracking, Pathfinder has to add additional possible paths at all of the mentioned nodes. But since it uses best-first search, these paths will never be explored unless all of the already expanded paths are less likely, so in most cases there is a minimum amount of additional work involved in adding these paths.

5.8.5 Improving the Efficiency of Ill-Formed Paths

Pathfinder can use two methods to overcome the inherent difficulty involved with ill-formed paths. The first method is to restrict the number of additional paths by using intelligent filtering in the successor function, such as the method for generating "reasonable" substituted steps described above. Although this will violate the admissibility property of the algorithm in theory, since not all possible successors are being generated, it is a highly reliable method in practice.

The other method is to exploit a property of the classification of plan-based misconceptions. Section 4.3.2 mentioned the possibility of a particular misconception being classified in two different ways, depending on the structure of the plan hierarchy. The most common ambiguity is between violated bindings and substituted steps. And it

happens that substituted steps are structural misconceptions, which are difficult to detect, while violated bindings are constraint misconceptions, which are much easier to detect. So if it is possible to design the hierarchy in such a way as to minimize the number of potential substituted steps, this can make the algorithm run more efficiently.

There is actually an entire spectrum of design possibilities. At one end of the spectrum, there are no bindings at all. Each step that would normally have bindings will be exploded into a different step for every possible combination of bindings. This will cause former violated bindings to show up as substituted steps instead. At the opposite end of the spectrum, all of the semantic significance of steps is transferred to the bindings, so that steps themselves are rather generic with a very large number of parameters. Now misconceptions that were previously substituted steps will show up as violated bindings.

Both of these extremes are ridiculously impractical. But they do illustrate two different trends that could be favored during the knowledge engineering process. In order to maximize the efficiency of Pathfinder, the knowledge engineer should attempt to minimize the number of similar steps in the hierarchy. If two steps that accomplish essentially the same task can be combined by introducing an extra parameter, then this should be preferred to keeping them distinct. This will cause the ambiguous misconceptions to tend toward the easier violated bindings instead of the harder substituted steps, and will reduce the number of ill-formed paths that Pathfinder considers.

Remember also that just because Pathfinder generates a path does not mean that the path will actually be explored. Assuming that the initial estimates of the probabilities are reasonably accurate, only the most likely paths will be verified.

5.8.6 Special Cases: Mentioned Steps and Preconditions

There are three types of special queries that are handled by Pathfinder. The first type is a **mentioned precondition**. Here, the user explicitly states that a precondition is (or is not) required for a plan. Recall from Example 8 on page 55 that this is the only way for an extra precondition to be detected. In this case, the last mentioned step in the user's query is actually a mentioned precondition, and the second-to-last mentioned step is treated as the queried step. Once the search reaches the queried step, Pathfinder checks to see if the mentioned precondition is required for the queried step. It then decides whether this constitutes a misconception, based on the user's claim and

based on whether the precondition (or a suitably similar precondition) was found.

The second type of special query is a **mentioned missing step**. In this case, the user explicitly says that a particular step is *not* part of the current plan. This is treated similarly to a mentioned precondition. The second-to-last mentioned step is treated as the queried step, and Pathfinder checks to see if the mentioned missing step is a descendant of the queried step. If it is, then the user is missing a step in his plan. Otherwise, no misconception has occurred.

The final type of special query is a **mentioned sequence**. Here, rather than having a single queried step, the user is querying an entire sequence of steps. He is basically describing the entire subplan for achieving a particular plan. Pathfinder currently only handles one special case of this type of query, namely the case where the user is listing all substeps of the last mentioned step. In this one case, Pathfinder retrieves the known substeps of the last queried step, and compares this sequence with the user's version. If all steps match correctly, then there are no misconceptions. Otherwise, the sequence is ill-formed and Pathfinder finds the closest match for each of the steps in the user's sequence. Imperfect matches are substituted steps; steps for which there are no matches are extra steps; and any leftover steps in the correct sequence are missing steps. Finally, any steps that are out of sequence signify violated orderings.

Handling general cases of mentioned sequences is more complex, since the steps that the user is mentioning may fall at various levels in the hierarchy. This involves proving that the user's collection of steps comprise a complete plan for the last mentioned step. It should be possible to handle this by adapting the method for constructing the table of completed steps described on page 86, but this is not implemented in the current version of Pathfinder.

5.9 A Demonstration

This section simulates the Pathfinder algorithm running on the query from Example 2:

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

It uses the hierarchy in Figure 5.8, which is adapted from Figure 1.1. The only change is to shorten the hierarchy to focus on the interesting steps relevant to the query. Although the actual program uses probabilities for the heuristics, this demonstration just shows a relative ranking of the various alternatives to simplify matters.

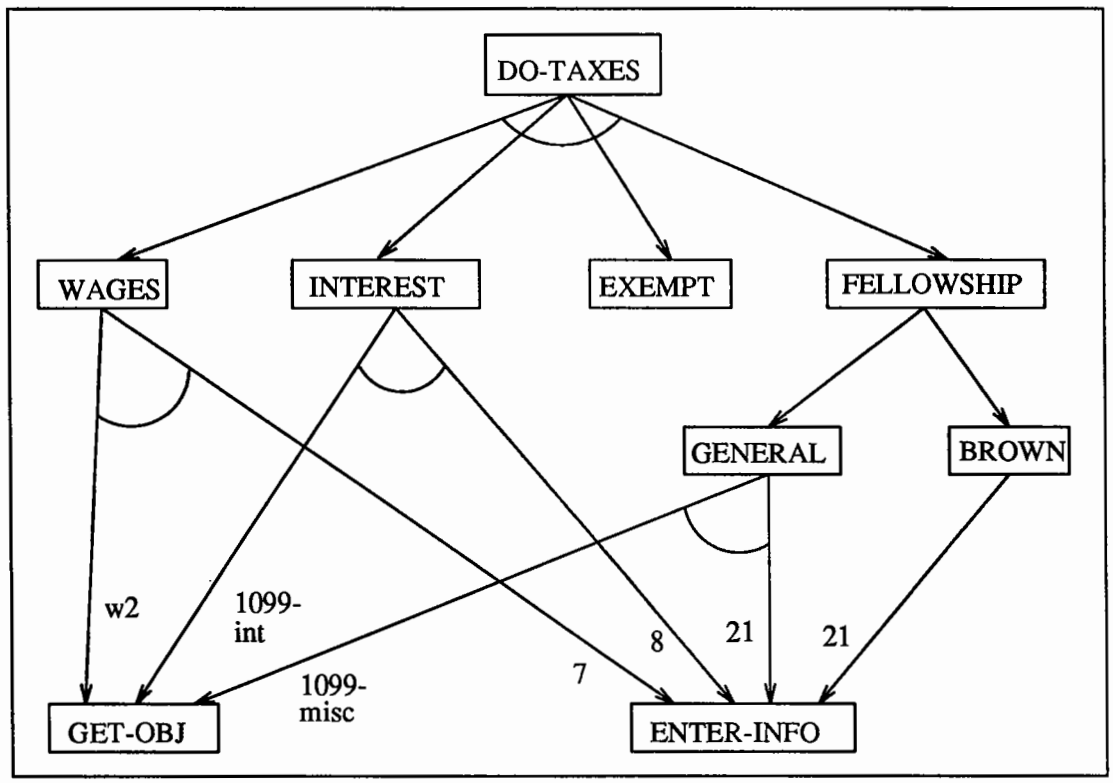

Figure 5.8: A simplified hierarchy.

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. In this case, there are two reasonable substitutions for filling in fellowship information: fill-in-wages (because fellowships are similar to certain types of wages) and fill-in-exempt-income (because fellowships were exempt at one time). But there is no path connecting fill-in-exempt-income with get-obj, so this path is discarded. The rationality 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.

Substituting fill-in-wages and fill-in-fellowship is acceptable, since there is a path from do-taxes through fill-in-wages to get-obj. So we now have two partial paths:

|| do-taxes → fellowship

|| do-taxes → wages/fell → get-obj.

The path for filling in fellowship information is obviously preferred at this point, since the only other option proposes a substituted step when we have 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 this path is still preferred to the path with fill-in-wages.

Now Pathfinder expands fill-in-fellowship down to get-obj. There is only one way to do this in the hierarchy, by passing through fill-in-general-fellowship. Since this is a very likely path, 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

|| do-taxes → wages/fell → get-obj.

Since we still have not encountered any misconceptions, this path is still preferred to the one with `fill-in-wages`. So Pathfinder will verify the rest of this path. Now 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. 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 lowers the likelihood of this path, and since it involves a misconception, Pathfinder tries to find another possible path connecting `fill-in-fellowship` and `get-obj` that avoids this misconception. There is another possibility; the user might be trying to go through `fill-in-brown-fellowship`, which would avoid the misconceptions, but which would require an extra step. Assuming that the missing precondition explanation is still preferred, this gives:

do-taxes → fellowship → general* → get-obj* ||

do-taxes → fellowship ||→ brown → get-obj+

|| do-taxes → wages/fell → get-obj.

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

When Pathfinder is actually run on this example, it does in fact prefer the explanation that passes through `fill-in-general-fellowship`. It postulates that the user is missing the precondition about fellowship types, and that he has a violated binding on the form that he is supposed to get. This explanation is only slightly preferred over the other two, so the response generator would probably want to phrase the response in such a way as to correct all three possible misconceptions[4].

The results of this simulation are summarized in Figure 5.9.

[4]The current probabilities that are assigned to the three explanations are 0.48 for the missing precondition and violated binding explanation, 0.41 for the extra step explanation, and 0.11 for the substituted step explanation.

	NOTES	GENERATED	OPEN	CLOSED
0	Start		T	
1	Expand T	TF $T^{W}/_{F}$ O		T
2	Verify TF	$T^{W}/_{F}$ O	TF	T
3	Expand TF	FGO FBO^{+} $T^{W}/_{F}$ O		T TF
4	Verify FGO	FBO^{+} $T^{W}/_{F}$ O	FG*O*	T TF
5	Exit TFG*O	FBO^{+} $T^{W}/_{F}$ O		T TF FG*O*

T = do-taxes G = fill-in-general-fellowship
W = fill-in-wages B = fill-in-brown-fellowship
F = fill-in-fellowship O = get-obj

$^{X}/_{Y}$ = substituted step
X^{+} = extra step
X^{*} = constraint misconception

Figure 5.9: Simulation of Pathfinder on the fellowship example.

5.10 Runtime Analysis of Pathfinder

This section provides a best-case and worst-case analysis of the Pathfinder program. The analyses of the three other algorithms in Section 5.7 measured the total number of nodes expanded. But since the expansion process is broken into two pieces in Pathfinder, and since the **Verify** procedure is more costly than generating the successors, we will measure the performance of Pathfinder by the total number of nodes that are verified by the algorithm. The following notation is used:

G = graph (plan hierarchy)
T = tree corresponding to G with duplication of nodes
m = number of mentioned nodes (not counting the root)
n_i = mentioned node in G (and in T)
b = backward branching factor (ambiguity) of G
f = forward branching factor of G (and of T)
d = depth of G (and of T).

The analysis can be made much simpler with the following assumptions:

- There is one unique root, n_0.
- The search starts at n_0.
- The queried step n_m is at a leaf.
- All leaves are at the same depth (namely, d).
- The branching factors b and f are constant for the entire graph.
- Mentioned nodes are evenly spaced along the path (at a distance of d/m).

In the typical graph searching problems, all important attributes are located at the nodes of the graph. The only values that might be found on the arcs are weights that are used by the heuristic function. In contrast, the plan hierarchy also has attribute information stored on the arcs. For example, binding constraints for a step need to be stored on the arcs because this information may change depending on which plan invokes the step. So instead of performing the analysis on the graph G, it makes more sense to refer to the tree T, which is formed from G by making a separate copy of a node for every incoming arc. This increases the number of nodes, and reduces the

backward branching factor b to 1. But it does not affect the forward branching factor f, nor does it affect the depth d.

5.10.1 Worst-Case Analysis of Pathfinder

In the type of plan recognition that Pathfinder does, the search is constrained to pass through each mentioned node. This is very helpful because the search problem can now be decomposed in such a way that an optimal solution to the entire problem can be constructed from optimal solutions to the subproblems. In other words, the optimal path from the root to n_m is constructed from optimal subpaths between each of the consecutive mentioned nodes.

For the moment, assume that there are no ill-formed misconceptions; all mentioned steps can be found along a path that already exists in the hierarchy. Since Pathfinder has all of the partial paths pre-computed in a hash table, it can construct all well-formed paths between two consecutive mentioned steps directly without actually visiting any of the nodes. It will only visit a node during the verification process, and this is done when a partial path is selected by the **Verify** procedure.

In the worst case, all partial paths between n_i and n_{i+1} will be verified before any are extended to n_{i+2}. Then all will be extended to n_{i+2}, and all of these new extensions will be verified, until one of the extensions finally reaches n_m (the queried step). It may be the very last extension that we verify that ends up being the optimal one, so even at the bottom level we may need to visit all of the nodes in all of the partial paths.

Each partial path contains $\frac{d}{m} + 1$ nodes. But except for the very first path, the root of each partial path will have already been verified on the previous iteration, so it doesn't need to be verified again. This gives $\frac{d}{m}$ nodes that need to be verified along each partial path. There are $f^{\frac{d}{m}}$ partial paths of length $\frac{d}{m}$ starting at n_i, so since each partial path is verified separately, an upper bound on the number of verified nodes between n_i and n_{i+1} is $\frac{d}{m}(f^{\frac{d}{m}})$. This upper bound is only realized in the degenerate case where every partial path out of n_i passes through n_{i+1}.

Note that the combinatorial explosion is completely contained in the subtrees. No matter how many paths there are between two mentioned steps, only one of them (the one judged to be the most optimal) will be expanded, and it will be connected to the optimal path between the next two mentioned steps. So the worst case total cost for Pathfinder (not including ill-formed paths) is simply the number of nodes verified

between consecutive mentioned steps, times the number of mentioned steps (m), or

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

verified nodes. The leading 1 is the cost of visiting the root of T.

This makes sense, since when $m = 1$ (only the queried step is mentioned) this gives $1 + d(f^d)$ expanded nodes. This corresponds to a complete traversal of every path in the tree from the root to the leaves. And when $m = d$ (every step except the root is mentioned), this gives a worst case of $1 + fd$ expanded nodes. This can be interpreted that at every level of the tree, we will select one node, generate all children of that node, and move to the next level.

Now let us consider the possibility of ill-formed paths. Missing steps are only detectable in special cases at the queried step n_m, and they take no more time than searching for a correct queried step in the worst case. Extra steps are much cheaper than correct steps; since there is no partial path associated with an extra step, the only node that needs to be verified is the step itself. So they will not occur in a worst-case scenario, and the only additional possibility that we need to consider is the case of substituted steps.

Recall that substituted steps are proposed by finding appropriate substitutions for n_i and making sure that those substitutions lie along a path from n_{i-1} to n_{i+1}. In the worst case, every possible path from n_{i-1} to n_{i+1} that does not go through n_i itself can contain a suitable substitution. So the only paths that will *not* be generated during the search for substitutions are the ones that contain n_i, which will already have been generated during the well-formed phase. This means that we are effectively doubling the length of the partial paths; at each mentioned step n_{i-1} instead of finding all partial paths to n_i, we will find all partial paths to n_{i+1} whether they pass through n_i or not.

This gives a total of

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

nodes that Pathfinder will have to verify in the worst case, including ill-formed paths.

Note that this upper bound will never be reached in practice. It involves degenerate trees where all leaves are the same, it requires that the heuristic function provide no information at all about alternative paths, and it requires that all substituted steps are equally likely. The number of partial paths between two mentioned steps will be much smaller in any realistic hierarchy, and as long as the heuristic function tells us

anything useful, many of the paths will never be verified (or even fully extended) in the first place.

It is obvious from this worst case analysis that the mentioned steps play a critical role in the efficiency of Pathfinder. The additional information provided by these mentioned steps is used to constrain the exponential search to isolated partitions of the hierarchy. As m increases and more steps are mentioned, the exponent of the formula decreases, resulting in fewer expanded nodes. As a comparison, the traditional **A*** algorithm does not make use of any intermediate nodes at all. So in the worst case it will verify every node in the tree T, giving a complexity of

$$\frac{f^{d+1}+1}{f-1}$$

verified nodes. So as long as $m \geq 2$, Pathfinder will do just as well as **A*** in the worst case, and will be able to recognize invalid plans that **A*** cannot handle. Pathfinder's advantage comes from the fact that it can make use of the additional information supplied by the mentioned steps to constrain the search space. The more steps that are mentioned, the better Pathfinder does.

5.10.2 Best-Case Analysis of Pathfinder

In the best case, Pathfinder's heuristic will have perfect knowledge and it will only verify nodes that lie along the final path. Also, in the best case there will be no appropriate steps that can be substituted for anything along the final path, so ill-formed paths can be ignored. So between n_i and n_{i+1} we will only verify the one partial path that will form the final answer; then we will find all extensions to n_{i+2} but again will only verify steps along one of the paths. Note that in the ideal case we do not need to visit any nodes to select the optimal partial path, since an initial estimate of the heuristic can be pre-computed and stored in the hash table along with the path itself. If this initial estimate is perfectly accurate, its value will not decrease during verification, so the first path that Pathfinder selects to verify will remain the optimal choice. So including the root node, this gives us

$$1 + d$$

verified nodes for the best case complexity of Pathfinder.

5.11 Performance of Pathfinder

Currently, of the 79 original examples of plan-based misconceptions in the transcripts, Pathfinder has been run on 39. It has also been run on 19 variations on these examples (for instance, changing certain facts about the world, or changing the query slightly), for a total of 58 examples.

There are several reasons for not attempting all of the examples. Since Pathfinder cannot handle violated optimization at present, there was no point in trying any of the 13 examples which only involved optimization. Also, since the current implementation does not do any detailed reasoning about the add/delete lists, there were 3 examples involving misconceptions about atomic actions that were not attempted. There were also the three examples in Section 4.3.3 that did not fit the classification in the first place, as well as the very confusing case of violated bindings in Example 4.

This still leaves 20 examples unaccounted for. There were two primary reasons for not including these. One is that the knowledge representation used in Pathfinder is not rich enough to capture all of the details in the dialogues, which made it impossible to represent some critical information. The other reason was simply a matter of time. Designing a plan hierarchy properly is a tedious task, since there is a large amount of information that needs to be encoded in the plans and world knowledge for each individual topic. It was only profitable to encode all of the necessary information if there were several related examples that could use the same information.

Accuracy Results

Pathfinder performed remarkably well on the 58 examples that it attempted. In 46 cases (79% of the examples), it returned interpretations that are indistinguishable from what a human observer would conclude. 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 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, Example 17 (repeated here as Example 22) 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.

[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.

Example 22: Pathfinder does not assign reasonable probabilities.

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 completely believable numbers to the various possibilities. This is primarily because the methods for calculating the probabilities of misconceptions, which will be discussed in Chapter 6, are only partially implemented.

The only example where Pathfinder came up with the wrong explanation is Example 23. 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

[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.

Example 23: Pathfinder misses the Extra Step.

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.

This is admittedly not the best experimental methodology. In the ideal case, the classification and program should have been developed using one set of examples, and tested on a separate set. Unfortunately, we did not have a large enough sample set to do this (even though our set of examples is an order of magnitude larger than any previously collected). Therefore, in order to test both the classification and the program on a sufficient number of cases, roughly 50% of the examples were used during the development of the program itself, and the remainder were used to develop the probabilistic results. In approximately 30% of the cases, it was necessary to adjust the probabilities slightly to match the preferences that human experts have. The other 70% of the examples already had the correct preferences, and their probabilities did not need to be tailored.

Time Results

In addition to being highly accurate, the Pathfinder program also suggests that robust plan recognition can be automated in an efficient manner with current technology. Although the worst-case analysis of Pathfinder in Section 5.10 is alarmingly high,

Pathfinder does not even come close to this upper bound in real life.

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 seconds[5]. 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.

[5]The one example that took longer than 2 seconds was Example 23, which took 4.3 seconds.

Chapter 6

Probabilities as Heuristics

Abstract: *The heuristic function used to guide the best-first search in Chapter 5 can be implemented using probabilities. A probabilistic interpretation can be given for the perfect plan recognition problem from Chapter 3, but it needs to be modified to deal with plan-based misconceptions. The probabilities in this interpretation can be simplified in practice so that they only require a manageable set of numbers, which can be organized to form a model of the user.*

6.1 Introduction

Chapter 5 presented a series of algorithms for doing plan recognition based on the **A*** algorithm, culminating in the Pathfinder program. What was lacking in these methods was a clear specification of the evaluation function that can be used to guide the search. A probabilistic interpretation of plan recognition can provide a well-defined method for measuring the likelihood of competing explanations.

The basic idea is that the search heuristic uses the probability that the current path is the actual path that the agent is following. This heuristic is comprised of the probability of the agent selecting each step along the path, along with the probability of his having all of the misconceptions that have been proposed along the path. The numbers that make up this heuristic are readily available from information about the world and the user, and the heuristic can be calculated incrementally, using the probability of the partial path constructed so far as an overly optimistic estimate of the probability of the final complete path.

Section 6.2 develops this interpretation, starting with a probabilistic interpretation of perfect plan recognition and extending it to handle plan-based misconceptions. Section 6.3 shows how this interpretation can be simplified in practice, and describes a method for calculating the probability of a misconception by decomposing the probability into a set of features. Section 6.4 shows that we are justified in using these probabilities as a search heuristic. Section 6.5 discusses the actual calculations that Pathfinder performs for computing misconception scores. Finally, Section 6.6 suggests that the numbers that Pathfinder uses are reasonable to get in real life, and that it makes sense to organize these numbers in a user model.

6.2 A Probabilistic Interpretation of Plan Recognition

The plan recognition problem, as presented in Chapter 3, consists of finding the most likely vertical path through the plan hierarchy that accounts for all of the steps that the agent has mentioned. A probabilistic approach seems to be the obvious choice, since intuitively we want to choose the most "probable" path. Probabilities are also desirable because their properties are well-understood and because they have the potential for incremental monotonic updating, which makes them suitable for use in a best-first search.

This section starts off with a probabilistic interpretation of the Perfect Plan Recognition Problem, which is then generalized to cover flawed plans as well. It makes several assumptions, which are summarized along with the major results in Section 6.2.4. The following notation is used:

- lowercase letters (x, y, etc.) represent arbitrary nodes in the graph (or plans in the plan hierarchy). One node, r, is distinguished as the unique root of the hierarchy.
- M = set of mentioned nodes.
- S = set of all substep (AND) nodes.
- A = set of all alt (OR) nodes.
- W = state of the world.
- $\overline{xy}$ = arc from x to y. This arc has an attribute $\Pr(\overline{xy})$, as defined below.

- $\overline{\overline{r \wr x \wr}}$ = some member of the set of paths which start at r and pass through x. Similarly, $\overline{\overline{r \wr x}}$ is some member of the set of paths which start at r and end at x. We will use the convention that if the symbol "$\wr$" is used more than once in the same equation, it refers to the same subpath in each occurrence. If it is subscripted, as in "$\wr_i$", then each subscript denotes a unique subpath.
- nodes($\overline{\overline{r \wr x}}$) = the set of all nodes along the path $\overline{\overline{r \wr x}}$.

6.2.1 Perfect Plan Recognition

Using this notation, the Perfect Plan Recognition Problem can be restated as finding the shortest valid path $\overline{\overline{r \wr x}}$ which maximizes $\Pr(\overline{\overline{r \wr x}} \mid W, M)$ and contains all of the steps in M (i.e., $M \subseteq \text{nodes}(\overline{\overline{r \wr x}})$). Intuitively, we want the shortest path because there is no point in speculating about details that the agent did not include in his query.

It is possible to assign probabilities to the arcs of the plan hierarchy which measure the likelihood that an agent will choose a particular plan. The value stored on an arc $\overline{xy}$ from node x to node y is $\Pr(y \mid x, W)$, also written $\Pr(\overline{xy})$, which is the probability that the agent will select plan y given that he has already selected its parent x. This value takes into account all information about the state of the world W, but does *not* take into account the fact that the agent may have mentioned certain steps[1]. If $\overline{xy}$ is not part of a valid plan, then its value is 0. Otherwise, there are two cases. If y is a substep of x, this probability must be 1, since all substeps are required to be executed if the parent is chosen. But if y is an alt of x, then the probability measures the likelihood of y relative to the other possible ways to achieve x. This is consistent with the normal probabilistic interpretation of *AND/OR* graphs, where the product of the children of an *AND* node must be 1, and where the sum of the children of an *OR* node must be 1.

The probability that the user has selected a path $\overline{\overline{r \wr x y}}$ is defined to be the probability that he has selected all of the steps that make up that path:

$$\Pr(\overline{\overline{r \wr x y}}) \stackrel{\text{def}}{=} \Pr(r, \ldots, x, y).$$

To calculate the probability of a path, we need to make an independence assumption: once the parent of a step is fixed, knowledge about earlier ancestors does not affect the probability of selecting the step.

[1] For the moment, ignore the fact that these values are not generally available. This will be addressed in Section 6.3.1.

Assumption A1: If x is an ancestor of y and y is the parent of z, and if x, y and z all lie along a valid path, then

$$\Pr(z \mid x, y, Q, W) = \Pr(z \mid y, Q, W) \text{ for any other steps } Q.$$

This assumption is related to the Markov property described in [34, p. 372]. Here, Q includes any information we might know about other steps in the path (for instance, it would include any steps that the agent has mentioned).

We can now calculate the probability of the user choosing a particular path through the hierarchy recursively from the probabilities of its component arcs: the probability is simply equal to the product of the conditional probabilities of each step along the path:

$$\Pr(\overline{\overline{r \wr xy}} \mid W) = \Pr(\overline{\overline{r \wr x}} \mid W) \cdot \Pr(\overline{xy}).$$

Proof:

$$\begin{aligned}
\Pr(\overline{\overline{r \wr xy}} \mid W) &= \Pr(r, \ldots, x, y \mid W) && \text{by definition} \\
&= \Pr(r, \ldots, x \mid W) \cdot \Pr(y \mid r, \ldots, x, W) && \text{by chain rule [82, p. 31]} \\
&= \Pr(r, \ldots, x \mid W) \cdot \Pr(y \mid x, W) && \text{by A1} \\
&= \Pr(\overline{\overline{r \wr x}} \mid W) \cdot \Pr(y \mid x, W) && \text{by definition} \\
&= \Pr(\overline{\overline{r \wr x}} \mid W) \cdot \Pr(\overline{xy}) && \text{by definition.}
\end{aligned}$$

□

We will now extend these results to account for mentioned steps. The predicate "explains" takes a path, a state of the world, and a set of mentioned steps, and is true exactly when the path provides a minimal-length valid explanation for the mentioned steps. Specifically, it is true exactly when the path is valid, contains all of the mentioned steps, and is as short as possible (i.e., the most specific step in the path is the most specific mentioned step):

$$\text{explains}(\overline{\overline{r \wr x}}, W, M) \overset{\text{def}}{\iff} \begin{cases} \forall z \in \text{nodes}(\overline{\overline{r \wr x}}) : \text{the constraints of } z \text{ hold in } W \\ M \subseteq \text{nodes}(\overline{\overline{r \wr x}}) \\ x \in M. \end{cases}$$

In order to guarantee that the path truly is as short as possible, we need to assume that the plan hierarchy is acyclic; a step cannot be defined in terms of itself (even if it uses different bindings). In other words, a step cannot have an ancestor (or descendant) of the same name.

Assumption A2: The plan hierarchy is acyclic.

The reason for requiring a minimal path is that we are interested in the most general path that contains the mentioned steps. We do not want to speculate about possible specializations of the plan if the user does not mention anything more specific, since this could only serve to lower the confidence in the path.

Since an "explanatory" path must contain all of the mentioned steps M and must also have a minimal length, a corollary of this definition is that if a path "explains" the mentioned steps, then no subset *or* extension of the path will qualify as explanatory:

$$\text{explains}(\overline{\overline{r l x y}}, W, M) \implies \neg\text{explains}(\overline{\overline{r l x}}, W, M) \wedge \forall z : \neg\text{explains}(\overline{\overline{r l x y z}}, W, M).$$

We need two more assumptions: one guarantees that the minimal explanations cover all possible situations, and one says that the relative probability of explanatory paths does not actually have to be conditioned on the mentioned steps.

Assumption A3: The probabilities of all minimal explanations of M in W are required to sum to 1:

$$\sum_i \Pr(\overline{\overline{r l_i}} \mid W, M) : \text{explains}(\overline{\overline{r l_i}}, W, M) \;=\; 1.$$

Assumption A4: The relative probability of two paths that each explain M is independent of whether M is actually mentioned. Specifically, if $\overline{\overline{r l_1 x}}$ and $\overline{\overline{r l_2 y}}$ are each valid minimal explanations of M in W, then

$$\frac{\Pr(\overline{\overline{r l_1 x}} \mid W, M)}{\Pr(\overline{\overline{r l_2 y}} \mid W, M)} = \frac{\Pr(\overline{\overline{r l_1 x}} \mid W)}{\Pr(\overline{\overline{r l_2 y}} \mid W)}.$$

It is important to realize that Assumption A4 does *not* hold for the partial paths: if $\text{explains}(\overline{\overline{r l_1 v l_2 x}}, M, W)$ and $\text{explains}(\overline{\overline{r l_3 w l_4 x}}, M, W)$, then it is *not* necessarily true that

$$\frac{\Pr(\overline{\overline{r l_1 v}} \mid W, M)}{\Pr(\overline{\overline{r l_3 w}} \mid W, M)} = \frac{\Pr(\overline{\overline{r l_1 v}} \mid W)}{\Pr(\overline{\overline{r l_3 w}} \mid W)}.$$

This is because knowing that x must be part of the path might increase or decrease the probability of passing through v. For example, if we have the simple hierarchy shown in Figure 6.1, and if $M = \{x\}$, then we have:

$$\frac{\Pr(\overline{\overline{r v}} \mid W, M)}{\Pr(\overline{\overline{r w}} \mid W, M)} = \frac{\frac{0.6*0.01}{(0.6*0.01)+(0.4*0.99)}}{\frac{0.4*0.99}{(0.6*0.01)+(0.4*0.99)}} = 0.015$$

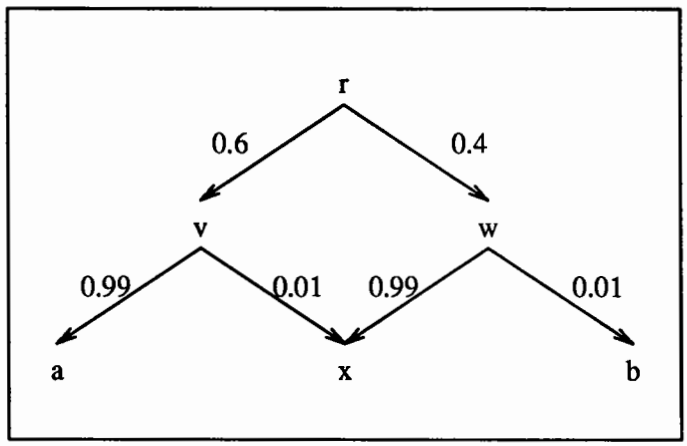

Figure 6.1: Knowledge of x affects the relative probability of $\overline{\overline{rv}}$.

But:

$$\frac{\Pr(\overline{\overline{rv}} \mid W, M)}{\Pr(\overline{\overline{rw}} \mid W, M)} = \frac{0.6}{0.4} = 1.5$$

So Assumption A4 only holds for the explanatory paths, which are not subject to this "extra" information because they already contain all of the mentioned steps.

We can now show that

$$\Pr(\overline{\overline{rlx}} \mid W, M) = \frac{\Pr(\overline{\overline{rlx}} \mid W)}{\sum_i \Pr(\overline{\overline{rl_i}} \mid W) : \text{explains}(\overline{\overline{rl_i}}, W, M)} \quad \text{if explains}(\overline{\overline{rlx}}, W, M).$$

Proof: assume that explains$(\overline{\overline{rlx}} \mid W, M)$.

$$\begin{aligned}
\Pr(\overline{\overline{rlx}} \mid W, M) &= 1 - \sum_i \Pr(\overline{\overline{rl_i}} \mid W, M) : \text{explains}(\overline{\overline{rl_i}}, W, M),\ \overline{\overline{rl_i}} \neq \overline{\overline{rlx}} \quad \text{by A3} \\
&= 1 + \Pr(\overline{\overline{rlx}} \mid W, M) \\
&\quad - \sum_i \Pr(\overline{\overline{rl_i}} \mid W, M) : \text{explains}(\overline{\overline{rl_i}}, W, M) \\
&= 1 + \Pr(\overline{\overline{rlx}} \mid W, M) \\
&\quad - \sum_i \frac{\Pr(\overline{\overline{rlx}} \mid W, M) \cdot \Pr(\overline{\overline{rl_i}} \mid W)}{\Pr(\overline{\overline{rlx}} \mid W)} : \text{explains}(\overline{\overline{rl_i}}, W, M) \quad \text{by A4} \\
&= 1 + \Pr(\overline{\overline{rlx}} \mid W, M) \\
&\quad - \frac{\Pr(\overline{\overline{rlx}} \mid W, M)}{\Pr(\overline{\overline{rlx}} \mid W)} \sum_i \Pr(\overline{\overline{rl_i}} \mid W) : \text{explains}(\overline{\overline{rl_i}}, W, M).
\end{aligned}$$

After simplification,

$$\Pr(\overline{\overline{r \lfloor x}} \mid W, M) = \frac{\Pr(\overline{\overline{r \lfloor x}} \mid W)}{\sum_i \Pr(\overline{\overline{r \lfloor_i}} \mid W) : \text{explains}(\overline{\overline{r \lfloor_i}}, W, M)} \quad \text{if explains}(\overline{\overline{r \lfloor x}}, W, M). \quad \square$$

Thus, we can ignore the effects of mentioned steps during the calculation of a path, and just normalize over all "explanatory" paths at the end. This is exactly the result we need for the Perfect Plan Recognition Problem, which asks for the path $\overline{\overline{r \lfloor x}}$ that maximizes $\Pr(\overline{\overline{r \lfloor x}} \mid W, M)$ and satisfies the constraint "explains$(\overline{\overline{r \lfloor x}}, W, M)$".

6.2.2 Violated Plan Recognition

This view of probabilities works fine if the agent always has correct plans. But it does not account for the cases where the agent's plan is flawed. If we take exactly the same interpretation as in perfect plan recognition, then the probability of an arc $\overline{xy}$ is the probability of the agent selecting this particular step y, given that he has already selected its parent x, *and* given that the agent has no misconceptions about y. Instead, to handle violated plans, $\overline{xy}$ is now interpreted to be the probability of selecting y given x, *regardless* of any violated misconceptions at y.

Unfortunately, there are an infinite number of potential misconceptions that could occur at y. To see this, we only need to consider a single violated binding; the agent could conceivably replace the correct binding with any value at all. Also, since it is possible to have multiple misconceptions at a single node, we need to consider all possible *combinations* of the misconceptions that could occur at y. There may be several ways to explain the agent's mistake, using different combinations of misconceptions. And because some of these explanations may be more likely than others, the misconceptions somehow have to be reflected in the hierarchy in order to facilitate selecting the most probable path.

This is done by changing the interpretation of a node. Each node in the original hierarchy is replaced by a subgraph containing an infinite number of "interior" nodes. For each node y in the original hierarchy, there is now one interior node for every possible combination of misconceptions that could occur at y. These interior nodes will be designated using subscripts. y_1 is distinguished as the interior node which corresponds to step y with no misconceptions at all. Some node y_i might correspond to the case where the agent has misconceptions a and b, but not misconception c. All

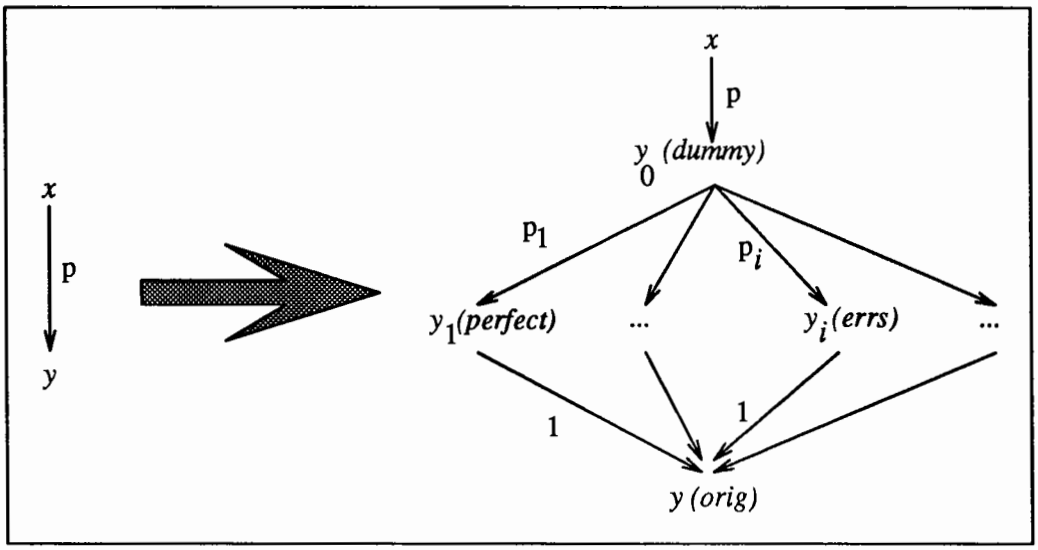

Figure 6.2: The transformation of $\overline{xy}$ for violated misconceptions.

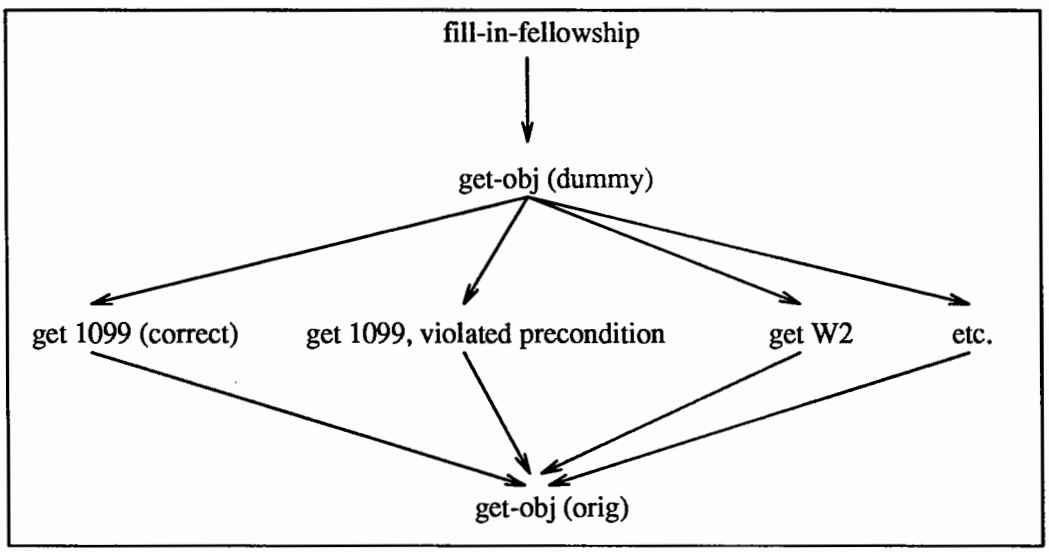

Figure 6.3: An example of a specific transformation for violated misconceptions.

of these new nodes are joined as children of a single *OR* node y_0 (which retains the original connection to x), and the relative probabilities of the new children are stored on the arcs. They also all feed into the "original" y node with single substep arcs having probabilities of 1. This is shown in Figure 6.2, with a specific example illustrated in Figure 6.3.

This transformation of nodes changes the method for calculating the probability of

a step given its parent. The original probability of y given x is now stored on the arc $\overline{xy_0}$, but it is no longer certain that y is a correct plan (i.e., that $\Pr(\overline{y_0 y_1}) = 1$). This probability $\Pr(\overline{y_0 y_1})$ is still typically quite close to 1, but there are now other violated possibilities that need to be included. Technically, the probability of $\overline{xy}$ should now be computed like the probability of a path, because y has been expanded into a subgraph. Since this subgraph only has a depth of 2, the result is the probability of $\overline{xy_0}$ times all ways of getting from y_0 to y:

$$\Pr(\overline{xy}) = \Pr(\overline{xy_0}) \cdot \sum_i \Pr(\overline{y_0 y_i}).$$

Note that we can ignore the $\Pr(\overline{y_i y})$ since they are all 1. Even though $\Pr(\overline{xy})$ is now represented differently, it still evaluates to its initial value, since the summation at the end of the equation will always equal 1.

The formulas of the last section still work exactly as before, except that instead of requiring all constraints to hold at the nodes, we instead require that the constraints are consistent with any violations that have already been postulated. For example, if we have already proposed a violated precondition at y, then only paths which account for this violation can be included.

$$\text{consistent}(\overline{\overline{r \wr x}}, W) \stackrel{\text{def}}{\iff} \forall z \in \text{nodes}(\overline{\overline{r \wr x}}) : \quad \begin{array}{l}\text{the constraints of } z \text{ are consistent with} \\ \text{any violations postulated for } \overline{\overline{r \wr x}}, \\ \text{relative to } W.\end{array}$$

The definition for "explains" needs to be updated to account for this change:

$$\text{explains}(\overline{\overline{r \wr x}}, W, M) \stackrel{\text{def}}{\iff} \text{consistent}(\overline{\overline{r \wr x}}) \wedge M \subseteq \text{nodes}(\overline{\overline{r \wr x}}) \wedge x \in M.$$

Using this new version of "explains", the probability of a path given a set of mentioned steps is exactly as before:

$$\Pr(\overline{\overline{r \wr x}} \mid W, M) = \frac{\Pr(\overline{\overline{r \wr x}} \mid W)}{\sum_i \Pr(\overline{\overline{r \wr_i}} \mid W) : \text{explains}(\overline{\overline{r \wr_i}}, W, M)} \quad \text{if explains}(\overline{\overline{r \wr x}}, W, M).$$

So once the hierarchy has been transformed to include the new "interior" nodes, and once we take consistent violations into account, the conditional probability of a path given the mentioned steps of the agent's query can be calculated identically for both valid and violated plans.

6.2.3 Ill-Formed Plan Recognition

Recall that there are three types of ill-formed misconceptions that need to be handled: extra steps, substituted steps, and missing steps. Each of the three types of ill-formed plans is addressed separately.

To handle extra steps, we need to add new arcs to the hierarchy that connect any potential extra steps to their parents. To distinguish these new arcs from the original ones, each arc will now have another attribute which specifies whether or not it is well-formed (i.e., whether the arc is part of the correct hierarchy). Then the hierarchy is expanded by adding ill-formed arcs which represent extra steps. The probability on an ill-formed arc $\overline{xy}$ measures the likelihood that an agent would incorrectly include y as an extra step under x.

It is conceivable for the agent to include any node as an extra step under any other node. Since there is no way of knowing exactly which extra steps the agent actually has, the new hierarchy must be strongly connected[2]. This is fine for alt children; the *OR* node simply has an arc leading to every other node in the hierarchy, with appropriate probabilities on the arcs. But there is a problem with substeps: all probabilities for substeps of a plan are supposed to be 1. And we certainly do not want to claim that the probability of an extra substep is always 1 by definition. So instead, we will add a dummy *OR* node as a substep, and attach all the extra steps to this node (along with a "no-op" alternative to handle the case where the agent has no extra steps at all) as shown in Figure 6.4. This new subgraph can now legitimately get a probability of 1, thus preserving the consistent interpretation of the *AND/OR* graph.

Substituted steps can be handled by using a method similar to the one for violated misconceptions. Each original node y already has an associated subgraph of "interior" nodes in the new hierarchy which accounts for all combinations of all possible violations. Now, we will add a new interior node for each possible substituted step, which specifies that the agent should have done y but did instead z (for every possible substitution z). Attached to these nodes are subgraphs which specify all of the possible combinations of misconceptions that could occur at z. So there may now be an interior node y/z_j which represents the situation where the agent mistakenly did z instead of y, and also has violated misconceptions a and b. This is shown in Figure 6.5.

[2]Technically, it is not quite right to make the hierarchy strongly connected, since we still do not allow cycles. In practice, this can be handled by assigning a probability of 0 to any new arc whose addition would create a cycle.

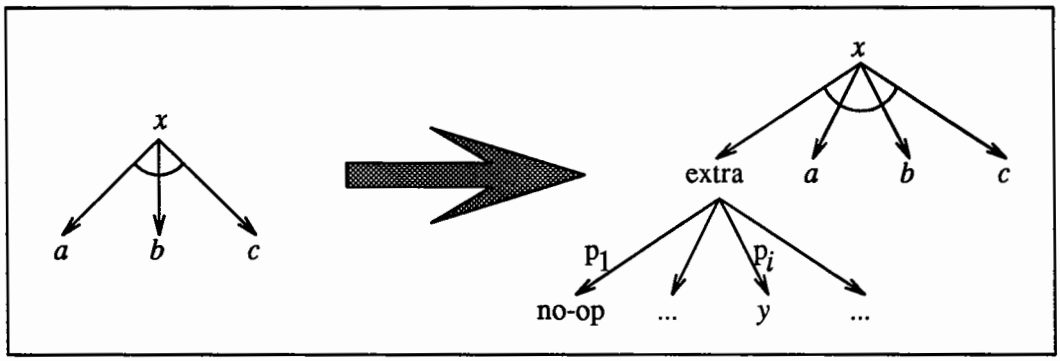

Figure 6.4: The transformation for an extra step y under x.

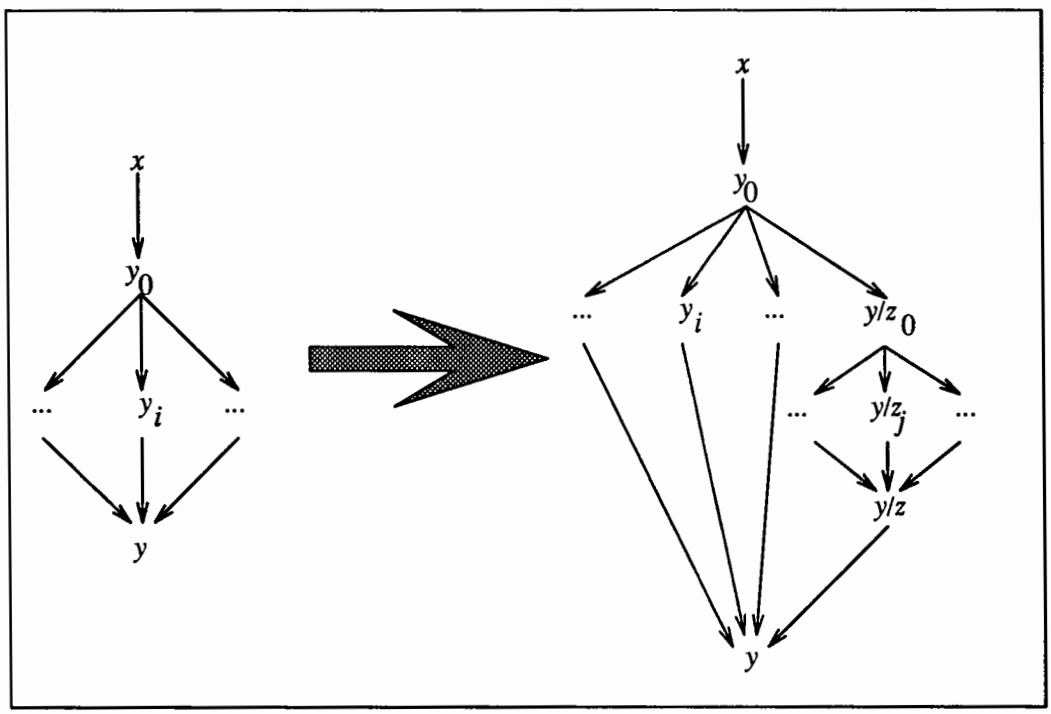

Figure 6.5: The transformation for a substituted step z at y.

Missing steps can be handled in the same way by adding another interior node, representing the case where the agent should have done y but instead did nothing at all (because y was missing from his plan for x). This can be interpreted as a slightly odd substituted step, where the incorrect step that is being "substituted" is vacuous.

Once the plan hierarchy is transformed to include all of these new nodes and arcs, all of the formulas of Section 6.2.2 will now work with ill-formed plans as well, without modification. Once we have determined whether or not a node of the plan hierarchy is consistent with the current situation, the particular interpretation of the node is irrelevant as far as the calculations are concerned. We can view these new "interior" nodes as simply another layer of steps added to the plan hierarchy. The fact that they happen to represent invalid steps and that they require the user to have certain misconceptions does not affect the probabilistic manipulations at all.

6.2.4 Summary

The results of Section 6.2 are summarized in the following assumptions, definitions and equations, which hold for valid, violated, and ill-formed plans:

Assumption A1: If x is an ancestor of y and y is the parent of z, and if x, y and z all lie along a valid path, then

$$\Pr(z \mid x, y, Q, W) = \Pr(z \mid y, Q, W) \text{ for any other steps } Q.$$

Assumption A2: The plan hierarchy is acyclic.

Assumption A3: The probabilities of all minimal explanations of M in W are required to sum to 1:

$$\sum_i \Pr(\overline{\overline{rl_i}} \mid W, M) : \text{explains}(\overline{\overline{rl_i}}, W, M) \;=\; 1.$$

Assumption A4: The relative probability of two paths that each explain M is independent of whether M is actually mentioned. Specifically, if $\overline{\overline{rl_1x}}$ and $\overline{\overline{rl_2y}}$ are each valid minimal explanations of M in W, then

$$\frac{\Pr(\overline{\overline{rl_1x}} \mid W, M)}{\Pr(\overline{\overline{rl_2y}} \mid W, M)} = \frac{\Pr(\overline{\overline{rl_1x}} \mid W)}{\Pr(\overline{\overline{rl_2y}} \mid W)}.$$

$$\text{consistent}(\overline{\overline{r \lfloor x}}, W) \stackrel{\text{def}}{\Longleftrightarrow} \forall z \in \text{nodes}(\overline{\overline{r \lfloor x}}): \text{ the constraints of } z \text{ are consistent with any violations postulated for } \overline{\overline{r \lfloor x}}, \text{ relative to } W \tag{6.1}$$

$$\text{explains}(\overline{\overline{r \lfloor x}}, W, M) \stackrel{\text{def}}{\Longleftrightarrow} \text{consistent}(\overline{\overline{r \lfloor x}}) \wedge M \subseteq \text{nodes}(\overline{\overline{r \lfloor x}}) \wedge x \in M \tag{6.2}$$

$$\Pr(\overline{\overline{r \lfloor xy}} \mid W) = \Pr(\overline{\overline{r \lfloor x}} \mid W) \cdot \Pr(\overline{xy}) \tag{6.3}$$

$$\Pr(\overline{\overline{r \lfloor x}} \mid W, M) = \frac{\Pr(\overline{\overline{r \lfloor x}} \mid W)}{\sum_i \Pr(\overline{\overline{r \lfloor i}} \mid W) : \text{explains}(\overline{\overline{r \lfloor i}}, W, M)} \quad \text{if explains}(\overline{\overline{r \lfloor x}}, W, M). \tag{6.4}$$

So, at least in theory, it is now possible to compute the probability of any path that the agent may take, regardless of the misconceptions that it might contain. This probability is simply the product of the probabilities of all the steps and misconceptions that form the path, suitably normalized, which agrees precisely with the intuitive description presented at the beginning of this chapter.

6.3 A More Practical Model of Probabilities

The view of probabilities developed in the previous section, while providing a clean interpretation of the probabilities of flawed plans, is far from being implementable. It is at once both too strong and too weak. On the one hand, it makes very strong independence assumptions about steps and misconceptions which will not hold up in real life. On the other hand, it proposes an infinite plan hierarchy, with no suggestions for how to calculate all of the necessary numbers that make up the probability of a path. This section addresses these issues by showing how a program such as Pathfinder might make some reasonable simplifications to arrive at a workable model that is more practical to use.

6.3.1 Handling the Probabilities on the Arcs

Recall that the probabilities on the arcs of the hierarchy are conditioned on the state of the world: $\Pr(\overline{xy}) = \Pr(y \mid x, W)$. This made the calculations for the paths very neat in theory; we simply needed to take the product of the values on all of the arcs along the path. But the problem with this is that it does not suggest how to get the values for the arcs themselves. The probabilities can change with the state of the world, so there is no way to have them stored directly in the permanent plan hierarchy. And even worse, we need to be able to decide which paths are inconsistent before we do the search, which is clearly impossible.

It turns out that there *is* a way to have fixed numbers on the arcs after all, if we make some rather strong assumptions. Instead of trying to condition everything on the state of the world and trying to judge whether paths are consistent ahead of time, we will take a very naive approach. We will assume that the relative probability of two valid alts is independent of the state of the world. Specifically, we make the analog of Assumption A4 for consistent alts:

> **Assumption A5:** If y and z are alt children of x, and if $\overline{xy}$ and $\overline{xz}$ can each be part of a consistent explanatory plan, then
>
> $$\frac{\Pr(y \mid x, W)}{\Pr(z \mid x, W)} = \frac{\Pr(y \mid x)}{\Pr(z \mid x)}.$$

If we also assume that every single path in the hierarchy is always consistent, then we can just deal directly with $\Pr(y \mid x)$ and ignore W completely, calculating the path probabilities just as before.

Of course, this second assumption is ridiculous. There will obviously be paths that are inconsistent with W. The only problem is that we won't know which paths are inconsistent until we encounter an inconsistency during the search. So when we follow a path down from the root, as soon as we encounter an arc $\overline{xy}$ that is inconsistent, we need to set the value of that arc (and all its descendants) to 0. This is because the probability of incorporating an inconsistent arc into a consistent path must be 0, and we only want paths that are consistent with the current state of the world.

At this point, we should propagate the changes through the hierarchy, since the fact that $\overline{xy}$ is inconsistent will change the probabilities of other nodes in the hierarchy. But there is an alternative to this complicated dynamic renormalization. Instead of

propagating the probability changes through the hierarchy every time we find an inconsistent path, we can simply discard that path and do a single renormalization at the end of the search. We will treat the "naive" values as if they are the true probabilities conditioned on W. When we find an inconsistent path, we will eliminate that path from the search, but will *not* renormalize the remaining probabilities. If we continue the search to find *all* of the consistent explanations, then we can renormalize at the very end to make sure that all of the explanatory paths sum to 1.

We still need to convince ourselves that this gives us the correct result. Instead of looking at individual nodes and arcs, let us just consider complete paths from the root to a leaf. If we assume that all paths are consistent, then the product of all "naive" values along a path will be precisely the probability of the agent choosing that path, since that is how we defined the probabilities on the arcs in the first place. We can imagine doing an exhaustive search of the entire hierarchy and discovering which paths are inconsistent. Since the probabilities of all the consistent paths are required to sum to 1, we must renormalize all of the remaining consistent paths[3]. This will give us the correct probability for each consistent path in the hierarchy, correctly conditioned on the state of the world.

But notice that during this redistribution phase, all consistent paths are affected by exactly the same amount. So it really amounts to a final step of multiplying all consistent paths by a constant factor in order to guarantee that everything sums to 1. This cannot affect the relative likelihoods of the consistent paths; if one consistent path is twice as likely as another, then it will still be twice as likely after the renormalization. So in fact, unless we are interested in the probability itself, we do not actually have to perform the exhaustive search and the renormalization. We can do a standard best-first search, and stop as soon as we get the first complete explanatory path. The number associated with this path is not legitimately a probability (because it has not been normalized yet), but it is perfectly acceptable for a best-first search heuristic, and can still be used to measure the relative likelihoods of the top answers. And we can still reconstruct the actual probability if it is really needed.

[3]This can also be viewed as distributing the probabilities from the inconsistent paths equally over the remaining paths.

6.3.2 Relaxing the Independence Assumptions

Two of the simplifying assumptions made in the previous sections are fairly restrictive independence assumptions about probabilities. Assumption A1 stated that the probability of an alt step does not depend on anything in the path except its parent. And Assumption A5 stated that the relative probability of two valid alts is independent of the state of the world. Both of these assumptions can be relaxed to account for dependencies that naturally occur in real life, which improves the precision of the probabilities while still keeping the calculations manageable.

Step Dependence

Claiming that the relative probabilities of valid alternatives are always constant is obviously an oversimplification. There could be many things that affect the selection of a 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. A more mundane example is an agent's plan for getting dressed in the morning. If we consider the probability of his choosing a sweater as opposed to a tank top, he might be 50 times more likely to choose the sweater during the winter. But we might get exactly the opposite ratio in the middle of July.

This is also noticeable if we look at steps that have already occurred. The probability of extending a sequence of steps should depend on the steps that have already taken place. For example, if we know that the agent has already selected a tank top, then he will be more likely to continue this plan by putting on a pair of shorts rather than a pair of heavy wool pants.

Relaxing this independence assumption for steps requires us to use more sophisticated probabilities. The single probabilities on the arcs would need to be replaced by probability distributions, which would be able to take other facts about the current state of the world into account. Pathfinder does not currently take advantage of this approach.

Misconception Interdependencies

If we apply Assumption A1 to the interior misconception nodes, it claims that the probability of a misconception depends only on its associated step, and is independent

of any ancestors. In particular, it claims that the probability of a misconception is independent of any misconceptions that were detected earlier in the path. This is obviously false; misconceptions can certainly be dependent on each other. But it turns out that it is possible to view the misconceptions as being mostly independent, except for several well-defined categories of dependencies. These dependencies can be recognized on a case-by-case basis, and the probabilities can be adjusted appropriately when a dependency arises.

We have isolated three cases where one misconception can be dependent on another misconception in the path. The first case, **vp-vp**, occurs when there are two violated preconditions in the path that test related conditions. A very simple example of this is when a step and one of its descendants have identical preconditions, which could arise because of redundancies in the plan hierarchy. While many of these **vp-vp** cases can be eliminated by careful construction of the hierarchy, sometimes it is not desirable to do so because it would make the hierarchy much more convoluted and unnatural. And sometimes it really is necessary to have apparently redundant preconditions, because steps can be reached by multiple paths which might determine the placement of the precondition.

There are also situations in which **vp-vp** pairs are only partially dependent. For example, there may be a precondition in the `fill-in-fellowship` plan that states that the object must be a fellowship. And the `fill-in-brown-fellowship` plan may have a more specific precondition that states that this fellowship is actually a Brown fellowship. If the first precondition is already violated, then the second one will certainly be violated as well. Pathfinder handles this by computing a weak form of implication. If one violated precondition is implied by an earlier violated precondition, then it is considered completely dependent, and the probability of the path is not affected by the later violation at all. If the later precondition implies the earlier one, then it is considered partially dependent, and the probability of the path is only lowered by the amount that was not already accounted for by the earlier violation.

A second type of interdependence is **vb-vb**, where a violated binding is related to an earlier violated binding. Many of these cases result from the "cascading bindings" described on page 87, where a single incorrect binding will be inherited by descendants to cause multiple errors. Pathfinder handles these cases by simply ignoring any violated bindings that have already been detected once, assuming that the second violation is completely dependent on the first. **vb-vb** pairs can also occur when one binding is

defined in terms of another. For example, the plan for filling in general fellowships has a single parameter `?x` for the fellowship, and has a substep

```
(get-obj 1099-form (university-of ?x))
```

which has a binding that uses `?x`. If the binding for `?x` has already been violated, then the binding in `get-obj` for the location `(university-of ?x)`, which is dependent on `?x`, may also be violated.

The final type of interdependence is **vb-vp**, where a violated precondition makes use of an earlier violated binding. The rationale here is that the agent will be testing the precondition using the wrong values in the first place, so he will be more likely to not know that the precondition does not hold. For instance, one precondition for the `get-obj` step is that the object the agent is getting must be at the initial location. If the binding for the object (or the location) is violated, then this precondition is more likely to be violated. Again, these can be treated as either partially or completely dependent misconceptions, depending on the situation.

It is easy to recognize these cases of interdependent misconceptions simply by keeping track of all violated bindings and violated preconditions that have already been encountered on a path. Using this information, the probability for the second misconception can be adjusted automatically to account for the dependency.

6.3.3 Simplifying the Interior Nodes

One large problem that has not been addressed yet is how to reason about the transformed hierarchy in practice. For perfect plan recognition, the only numbers that are required are the probabilities of a node given its parent; they are simply the relative likelihoods of the alternate ways of achieving a plan, and given the analysis in Section 6.3.1, they can be stored directly on the arcs of the plan hierarchy.

Unfortunately, the "interior" nodes that were proposed to handle violated plans are an entirely different story. They transform each node in the hierarchy into an infinite subgraph, and every new arc in the subgraph requires additional probabilistic information about the misconceptions that could occur there. It is obviously not possible to pre-compute these values. The important thing to realize, however, is that all but a handful of the various combinations of misconceptions will be so unlikely that their probabilities will be extremely close to 0.

So we can make a simplifying approximation and assume that, unless there is evidence to the contrary, the probability of the "perfect" interpretation $\Pr(\overline{y_0 y_1})$ (with no misconceptions) is 1. This will have the effect of slightly overestimating the value of an arc when there are no misconceptions, but for the best-first search method used by Pathfinder, overly optimistic heuristics are required anyway. The only time when the estimation will be significantly off is if there are very likely misconceptions that can occur at a node. In these cases, the misconceptions will be predictable enough that they can be isolated in their own separate nodes, forming a "bug list" of the most common mistakes. This is exactly the situation in which a bug list approach is appropriate: it can successfully weed out the most frequent mistakes that have a significantly high probability, and it does not attempt to solve the entire problem by predicting all of the misconceptions ahead of time.

None of the remaining interior nodes actually needs to be represented in the hierarchy itself, on the grounds that the prior probabilities are low enough to be safely ignored. When there is enough evidence to support one of these nodes, the node can be generated dynamically as needed. The cases of constraint misconceptions are trivial. For example, if Pathfinder discovers that two preconditions are violated during the process of checking the preconditions of a node y, it will automatically construct the interior node corresponding to y combined with the two violated preconditions.

The interior nodes corresponding to structural misconceptions are treated similarly. Pathfinder only proposes missing and extra step nodes if they are required by the query. It also generates substituted step nodes as needed, by finding only those steps that would make "reasonable" substitutions, as described on page 89. Intuitively, a step makes a "reasonable" substitution if it is similar enough to the correct step that a path containing the substitution is more likely than a path without it.

Thus, in actual practice, it is not necessary to do the transformation to the interior nodes. We can keep the original hierarchy and add only the misconception nodes that are required by the explanation.

6.3.4 Features Influencing Misconception Scores

Since Pathfinder is now constructing these 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.

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 itself[4]. Pathfinder currently uses 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 the transcripts and experiments with Pathfinder have suggested that these six features provide a surprisingly reliable estimate of the likelihood of the misconception.

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 might be considered ten times more important than the feature of complexity.

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

Similarity

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.

Preconditions actually have two different types of similarity that can be measured. For substituted preconditions, the important fact is how similar the two versions of the precondition are to each other. For violated preconditions, it is important to determine how close the precondition is to holding. While preconditions are usually treated as

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

having binary truth values (either true or false), it is sometimes possible to measure how "badly" a precondition fails. For instance, the users in Examples 24 and 25 ask exactly the same question and have problems with the same precondition, but the actual misconception is different in each case. Since the two times are so close in Example 24, it is more likely that the user just didn't know exactly what time it was, signifying a violated precondition. But in Example 25, the two times are much further apart, and it is more likely that the precondition is substituted (or missing) instead of violated.

[U] I'm going to call the IRS up right now and complain about these tax laws. What's their number?

[E] Well, the number is 1-800-IAM-POOR, but it's 5:05 pm and their office just closed.

Example 24: Violated Precondition (times very close).

[U] I'm going to call the IRS up right now and complain about these tax laws. What's their number?

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

Example 25: Substituted or Missing Precondition (times very different).

Obscurity

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. For example, if a particular tax plan has a very obscure technicality that only a trained expert would know about, then an average user might have it missing from his version of the plan. On the other hand, if there is an extremely common precondition that everyone is expected to know, then it would be much less believable that the user would be missing it.

Obscurity can often be used to distinguish between missing preconditions and violated or substituted preconditions. Usually, when the precondition is not mentioned in the query, all the outside observer will know is that the precondition should have been satisfied but hasn't been, and she will not know whether the agent is missing the precondition, has a different precondition, or thinks that the precondition actually holds. 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 a missing precondition is less believable, and it is more likely that it is violated or substituted.

Obscurity can also be used to a lesser extent with extra components. If an agent adds an extra component that is fairly common, it is often more believable than if he adds a very specialized, uncommon component.

Complexity

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).

Complexity can sometimes be used to recognize substituted preconditions. If a precondition is very complex, a reasonable explanation would be that the agent might have substituted a simpler version of the precondition which might in fact hold. If the precondition is relatively simple, it is more likely that the agent is either missing the precondition entirely, or has the correct precondition but thinks that it is actually satisfied.

Overgeneralization

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. For instance, if every type of income *except* fellowship income involves getting some type of form, the agent might reason as follows:

> "Well, I have to get a form for my wages, I have to get a form for my interest, I have to get a form for my pension plan... I must have to get some sort of form for my fellowship information too."

This would cause him to have an extra step in his plan for filling out fellowship information.

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.

Permanence

In addition to overgeneralizing on the structure of a plan, an agent may also overgeneralize on the temporal permanence of a plan. There are two ways that permanence can influence the probability of a misconception. The most common form affects preconditions. Certain preconditions are time-invariant; for a given set of bindings, the precondition will always have the same truth value. Other preconditions can change their truth value over time. It may be the case that a precondition for a plan has 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.

The other type of permanence arises when the knowledge base of plans is dynamic. For example, certain rules in the tax laws change from year to year. The agent may be using a set of plans that is out of date, and that should not be applied any more.

Discourse

Often, things that were said earlier in a dialogue 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.

6.3.5 Summary

Although the complete hierarchy for invalid plans is infinite in theory, we don't actually need to make the transformation in practice. We can generate exactly the interior nodes that are needed for a particular query, and the probabilities for the interior nodes can be constructed at runtime by decomposing them into a set of easily calculated features.

The earlier chapters of this thesis were primarily concerned with *what* types of mistakes people make with their plans. But the six features of plan-based misconceptions that were presented in this section are the beginning of a theory of *how* people make mistakes. There are various types of faulty reasoning that can lead to a plan-based misconception, such as overgeneralizing on a plan component or getting two similar components confused. This faulty reasoning can explain how (and also why) an agent ended up with a faulty plan.

The "true" underlying incorrect beliefs that cause a plan-based misconception are still normally unattainable. But the six misconception features bring us closer to this level by isolating the properties that affect the likelihood of a misconception. Not only do they allow us to select the most likely explanation for a user's faulty query; they also provide us with some justification for *why* this explanation is the best.

This theory is far from being complete, but it helps our immediate problem of how to measure misconceptions reasonably accurately, and it can hopefully offer a starting point for developing a more detailed understanding of the causes of plan-based misconceptions.

6.4 The Appropriateness of Probabilities for Best-First Heuristics

Although the probabilistic interpretation developed in this chapter is practical enough that a program such as Pathfinder can actually calculate the necessary probabilities, it is still not obvious that these probabilities are appropriate as a heuristic to guide a best-first search.

Since Pathfinder is doing a best-first search, it does not have access to an entire

path at once. It will only have a partial path, which extends from the root of the hierarchy through some (but not all) of the mentioned steps. During each iteration of the algorithm, another path fragment between two mentioned steps will be added to the answer. Pathfinder uses the probabilities calculated *so far* on the partial path as an estimation for the probability of the entire path. For instance, in the middle of processing the standard graduate student example from Section 5.9, Pathfinder will have constructed a partial path from `do-taxes` down to `fill-in-fellowship`. It will use the probability of that partial path as an estimate of the entire path down to `get-obj`. Once it continues to expand this path, it will realize that this estimate is too high, because of the misconceptions that will be discovered at the bottom. So it will revise its estimation down to take the new information into account.

Admissibility Requirements

In order to properly use a probabilistic heuristic function for a plan recognition algorithm based on **A***, the function must meet two admissibility requirements: it must be *optimistic*, and it must be *monotonic*. This means that it must always overestimate the probability of a path, and that the probability of a path must never increase.

We know that probabilities are always ≤ 1, and that the heuristic value of a partial path is simply the (un-normalized) product of the probabilities of all of the arcs that make up the path. So the monotonicity property can be proven by showing that once an arc is incorporated into the probability of a path, its contribution to the probability of the path never changes. There are only two ways that this contribution could conceivably change during the search: by eliminating the arc from the path (through backtracking in the search), or by recomputing the value on the arc. The first possibility can be dismissed immediately, since Pathfinder never backtracks. Once an arc is added to a path, it will always remain on the path. There is only one situation where the value on an arc might change, and that is when the probabilities are renormalized. But since Pathfinder defers normalization until after the search is completed, the values on the arc remain unchanged, and thus the probability heuristic is guaranteed to be monotonically decreasing.

Proving that the heuristic is optimistic is a much more difficult manner. The best we can do is show that the heuristic is always optimistic, *modulo* the accuracy of the weights on the arcs. Assuming that the arcs are accurate, the optimistic property

follows directly from the fact that the partial path is taken as an estimation of the entire path, and from the fact that the heuristic is monotonically decreasing[5]. Unfortunately, there is no way to guarantee the accuracy of the arcs, which depends primarily on the reliability of the information we have about the user. There are techniques for modifying an inadmissible A* heuristic to conform to the admissibility requirements [32, 81], but these techniques often result in a loss of precision. Assuming that sufficiently detailed information is available about the user, and assuming that the designers of the hierarchy are relatively forgiving in their penalization of questionable misconceptions, the accuracy issue should not be a major problem. We are not actually concerned with absolute optimality anyway. For instance, if the top three paths all have very close probabilities, we will need to construct an answer that addresses all three possibilities regardless of their relative rankings. So slight variations of optimality aren't critical, and for all practical purposes, as long as the values on the arcs are reasonably accurate, the probability function can be considered an admissible candidate for best-first search.

6.5 Pathfinder's Actual Probabilities

This section describes the actual method that Pathfinder uses to calculate the probability of a particular path. While it follows the view of probabilities that has been developed in the previous sections fairly closely, some disclaimers are necessary at this point. Since a definitive theory of misconception features is yet to be developed, we are not terribly concerned with implementing the six misconception features exactly as presented in Section 6.3.4 for every type of misconception. Instead, we are attempting to implement the features that are both the most useful and the easiest to handle, in order to demonstrate that this probabilistic approach really is feasible. Finally, any specific numbers and formulas that are mentioned in this section should be taken with a very large grain of salt; we have absolutely no justification for them at all except that they look reasonable and appear to work correctly. But since nobody has ever applied probabilistic techniques to the problem of recognizing faulty plans before, we have to start somewhere.

[5]Note that the heuristic is *not* an optimistic estimate of the *probability* of a path, since the path probability can still increase after normalization. But the normalization is not done until after the search is completed, and is not considered part of the heuristic function. So the heuristic function itself is optimistic; the value for a partial path is always less than or equal to the value for the entire path.

6.5.1 Calculating the Probability of a Path

As discussed in Section 6.3.3, Pathfinder does not actually transform the plan hierarchy to include the "interior" nodes. It only generates exactly the interior nodes that are needed to represent the misconceptions along a path. It starts at the root, with a probability of 1. When it adds an alt node to the path, it multiplies the probability by the value stored on the corresponding arc. Adding substeps does not affect the probability, since the value on the arc will just be 1. When it verifies the step to check for violations, it multiplies the path probability by the probability of each misconception that is encountered. These misconceptions are treated as independent, except for the three specific types of interdependencies introduced on page 123.

Extra and missing steps are proposed only when they are needed, and the only value needed for them is the probability of the misconception itself, which is multiplied into the path probability. For substituted steps, Pathfinder checks the subpaths at both the correct and incorrect step for misconceptions, and then multiplies in the probability of the substituted step misconception itself.

So in constructing a path, Pathfinder calculates exactly what it is supposed to: the product of the probabilities stored on each arc in the path, times the product of the probabilities of all the misconceptions along the path. At the end, Pathfinder normalizes the probabilities over just those paths that provide a consistent explanation of the user's query.

6.5.2 Calculating the Probability of a Misconception

Pathfinder constructs the misconception probabilities using some of the features described in Section 6.3.4. For each misconception, the relevant features are calculated independently and combined using a weighted average, because the importance of a feature is different for each misconception type. For example, if substituted preconditions use similarity, obscurity, and complexity with respective weights of 10, 1, and 1, then the probability for a substituted precondition x will be

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

If the misconception turns out to be dependent on an earlier misconception, then the probability is adjusted to reflect only the "new" amount that hasn't already been explained by the earlier misconception, using the methods described on page 123.

There are four sources for the scores for the misconception features. Some scores are just constants; all misconceptions will receive the same value for these features. Some scores are retrieved from static values stored directly in the plan hierarchy. Some involve functions based on the number of certain types of arcs in the plan hierarchy. The remaining scores are calculated from information stored in other knowledge bases.

There are three such knowledge bases, which are structured as weighted discrimination nets. There is a net for objects, which we will call the O-Net. This is 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. And there is a third net for predicates that are used in preconditions, called the P-Net. These nets are used primarily to calculate the similarity of two entries, by taking the product of the weights along a connecting path.

We will now describe exactly how the probabilities are calculated for each class of misconception.

Violated Bindings

Instead of scoring a violated binding directly, Pathfinder specifies a list of important (salient) conditions that must hold for that binding, similar to the **salient features** used by McCoy [69]. These are the properties that are important about the binding for the current plan. These salient properties are checked using the agent's incorrect binding, to see how many of them are satisfied. Both common and distinct properties of the binding can be checked in this way [109]. Any violated properties can then be scored as violated preconditions, and can be combined to form the probability of the violated binding within the context of the current plan.

Violated Preconditions

There are three features that Pathfinder uses for violated preconditions. The most important (with a weight of 10) is similarity.

Similarity in this case measures how close the precondition is to being true. There are two ways that Pathfinder measures this, depending on the predicate of the precondition. If the predicate is "isa" (or a statement of equality), then Pathfinder looks up the two arguments in the O-Net to find out how closely they are related[6]. Otherwise,

[6]It can also handle certain similarities between numbers, and can do a crude measure of similar

it searches the state of the world for all facts that involve the same predicate, and checks the corresponding arguments in the O-Net. If the arguments are closely related, then the precondition "almost" holds (i.e., the agent might believe that it actually does hold), and the similarity score is the product of the similarities of the arguments. Any remaining similarity contributions that are not affected by the state of the world (i.e., contributions that are properties of the precondition itself, that would lead a user to believe that the precondition is true when it is actually false) are merged into a single number that is stored directly with the precondition.

Two secondary features for violated preconditions (each with a weight of 1) are obscurity and complexity. Obscurity is also a single number stored directly with the precondition. Complexity is based on the intricacy of the precondition, measured by the number of clauses in conjunctions and disjunctions. As the complexity increases, the chances of a misconception also increase:

$$\text{For } n \text{ clauses, complexity} = \begin{cases} 0.9 & \text{if } n \geq 10 \\ 10^{n-10} & \text{otherwise.} \end{cases}$$

Substituted Preconditions

The main feature for substituted preconditions is similarity (with a weight of 10). This is the product of two numbers: the similarity of the two predicates (calculated from the P-Net), and the weighted average of the similarity of the corresponding arguments (calculated from the O-Net). Pathfinder can make suitable adjustments for negated predicates, mismatched conjunctions and disjunctions, and nested predicates.

There are also two secondary features (each with a weight of 1). These are obscurity and complexity, which use the same values that were used for violated preconditions.

Missing Preconditions

Missing preconditions also use the same obscurity and complexity scores, but they are weighted differently; obscurity now has a weight of 10, compared to complexity's weight of 1.

---

spellings (for example, to capture the similarity of "W2-form" and "W4-form").

Extra Preconditions

Extra preconditions currently do not use any features at all. They just get assigned a constant score of 0.001.

Violated Optimization

The current version of Pathfinder does not even detect violated optimization in the first place. So it only has a dummy score of 0.9.

Substituted Steps

The probability of a substituted step is comprised of three equally weighted features: similarity, obscurity, and complexity. Similarity is measured from the S-Net. Obscurity is stored directly on the arc of the plan hierarchy. And complexity is based on the number of substeps that the correct step has. If the correct step is an alt node, then it is based on the average number of substeps for each alt child.

$$\text{For } n \text{ substeps, complexity} = \begin{cases} 0.9 & \text{if } n \geq 5 \\ 10^{2n-10} & \text{otherwise} \end{cases}$$

Missing Steps

The probability of a missing steps uses a single feature of obscurity, which has the same stored value that is used for substituted steps.

Extra Steps

Extra steps use two equally weighted features: obscurity and overgeneralization. Obscurity is based on the total number of plans that correctly contain the extra step; it is the ratio of the number of possible paths that correctly contain the extra step to the total number of paths in the plan hierarchy.

Overgeneralization is based on only the number of *similar* plans that contain the extra step; Pathfinder finds all plans that it considers similar to the extra step's parent (from the S-Net). The overgeneralization score is the average of the similarity measures between the parent and each similar step that correctly contains the extra step[7].

[7]These features are obviously not independent, although they are assumed to be.

	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	—									

Figure 6.6: Misconception features currently used by Pathfinder.

Violated Orderings

Violated orderings just get a constant score of 0.9, since in most cases their presence does not appear to dramatically affect the believability of a path.

Summary

The specific features that Pathfinder currently uses for the various misconception types are summarized in Figure 6.6. A dash signifies that the corresponding feature is definitely inappropriate for that misconception type and will never be used. The blank entries signify that the feature has not been implemented yet, and may be appropriate for that misconception type. It is clear that even with this incomplete classification of misconception features, there are still several holes to be filled in.

6.6 Are the Numbers Manageable?

While Section 6.3 simplified the probabilistic interpretation to make it more computationally feasible and broke misconception scores down into a set of simpler features, it still did not address the issue of whether or not the required numbers are available. There are two types of numbers that are needed in order to calculate the likelihood of a path: the step probabilities that are stored on the arcs of the plan hierarchy, and the misconception features described earlier in this chapter. Since the probabilities might vary from person to person, it makes sense to organize them in a user model. This section suggests that it is feasible to get the necessary numbers in real life, and describes how a user model can help to maintain them.

6.6.1 The Role of User Models in Plan Recognition

User models are already employed quite frequently in natural language generation systems to decide how to phrase a response, since the same question asked by different people should get different answers [29]. This can be done in several different ways [52]. One common way is to use stereotypical models of "expert" and "novice" users, which can help to tailor a response to the correct level of expertise [79]. This can be generalized to allow a user to be an "expert" in certain areas and a "novice" in others [80]. The modeling process may also involve a larger set of stereotypes [71, 89], or it may attempt to tailor a model to one specific user [51].

But user modeling is also strongly tied to plan recognition. As we have already seen, even perfect plan recognition is frequently ambiguous. 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 [40]. 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 supermarket, or he may just be out for a stroll.

In most cases, however, we would immediately jump to the conclusion that Jack is going to the supermarket to do his shopping, and would not even consider the hundreds of other possible explanations. This sort of reasoning can be explained by using the information provided by a user model. If we aren't given any evidence to the contrary, we would assign Jack to some generic user model, which tells us that most people go to the store to do shopping. If we have heard that Jack is a known fugitive on the FBI's most wanted list, we might assign him to a "dangerous-felon" user model, which would prefer the explanation of robbing the store. Different user models will have different preferences for plans in the domain.

It is also well known that different types of people 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 [96]. Such tendencies toward certain types of mistakes should be reflected in the user model, similar to the way that the model reflects tendencies to prefer certain plans over others. These preferences for one plan over another, or for one mistake over another, are exactly

what is captured by the probabilities that are used by Pathfinder.

Notice that this does not specify whether the probabilities are for all users, for a stereotype class of users, or for one specific user. Since the model itself can be completely isolated and replaced, we could conceivably have a separate model tailored to every individual user, and when the system is reasoning about a particular user's actions, it will retrieve 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.

6.6.2 What Numbers are Needed?

Given the probabilistic view outlined in this chapter, we can determine exactly what numbers are needed. 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 parent[8].

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. 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.

- 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.

[8]Remember that for the case of substeps, the probability is always a constant 1.

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 itself[9], and the number of predicates in the P-Net will be negligible compared to the other two nets.

This is a very 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.

6.7 Summary

The actual probabilities that Pathfinder finally ends up using are quite different from the initial formulation in the beginning of this chapter. We started with a probabilistic interpretation of perfect plan recognition, which was then extended to handle plan-based misconceptions. It was fine in theory, but it made too many independence assumptions and it required an infinite plan hierarchy. We were then able to relax the independence assumptions, and at the same time we found a way to calculate the probabilities using the original hierarchy. This method turned out to still be admissible as a search heuristic, and the actual probabilities of the individual steps and misconceptions could be calculated from numbers that are reasonable to obtain in real life.

[9]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.

So given the classification of a plan-based misconception (which is provided for free during the task of detecting the misconception in the first place), and given a user model that will provide a way to measure the misconception features for a particular user, we now have the ability to calculate the probability of novel misconceptions. This, combined with the likelihood of individual steps, provides us with all of the numbers necessary to calculate the complete probability of a path in the presence of possible user misconceptions.

Chapter 7

Conclusions

Abstract: *This final chapter reflects on where this research started and on where it has come. It reviews the contributions that this thesis has made, and suggests some possible future directions for extending the work.*

7.1 Summary of Contributions

Virtually all of the plan recognition techniques of the past (and the present) have ignored the problem of human fallibility. They have assumed that people will always know what they are doing, will always have correct plans, and will always execute those plans properly. This creates a serious problem when systems that use these techniques are faced with a flawed plan; most of these systems are completely unable to understand what the agent is doing in these cases. Intelligent interfaces, intelligent tutoring systems, and any other systems that endeavor to interact intelligently with people need the capacity to recognize, diagnose, and correct the mistakes that are bound to occur.

This thesis has approached the problem from the perspective of extending traditional plan recognition methods to improve the reliability and helpfulness of intelligent interfaces. These intelligent interfaces are designed to interact with a wide range of users, many of whom are novices that do *not* always know what they are doing. These novices are exactly the people who can benefit most from the assistance of an intelligent interface, but this assistance breaks down at precisely the point where the novices need it the most.

System designers have typically dealt with this problem by including bug lists of

common mistakes, but this only helps if the user happens to have one of the misconceptions that the designer anticipated. There are many studies that have shown that bug lists are not robust enough to handle the full range of mistakes that people make. Instead of trying to predict these mistakes one at a time, it makes more sense to attack the problem at the source, by reasoning directly about the way that people construct plans. This has produced a new classification of plan-based misconceptions that covers all ways that a plan can be constructed or executed improperly. There are several critical advantages to this approach. Unlike some of the earlier, unprincipled classifications, this is based on the structure of plans and provides a complete classification of *all* types of plan-based misconceptions. It is also completely independent of the domain; the classification is not tied to a particular application (such as Pascal programs or data entry). Finally, since the classification is closely coupled with the plan structure, the detection of misconceptions can be combined with the plan recognition process itself.

Multiple possible interpretations of misconceptions also created difficulty. "Perfect" plan recognition already produces ambiguous plans quite frequently, but the ambiguity problem in flawed plans is much worse and the need for an accurate diagnosis is even greater. Since a user with a misconception is already confused, a bad response will only serve to make matters worse. Probabilistic methods can help by addressing the root of the problem. It is possible to isolate the primary "features" that determine whether or not a plan-based misconception is reasonable in a given situation. These features can be calculated from a model of the user, and once the class of misconception is known, the features can be combined to form the probability that the user has this misconception.

By recasting the plan recognition problem as a graph-searching problem, a best-first search method based on the **A*** algorithm can be used, which is able to find the most likely interpretation of the user's actions. Combining this with the ability to detect and weigh misconceptions has resulted in the Pathfinder program, which uses probabilistic reasoning to guide a search for the most reasonable plan that the user could be following.

Both the theoretical classification and the practical implementation have been tested on a large corpus of naturally occurring plan-based misconceptions, which were extracted from real conversations in a variety of domains. The classification has proven to be very reliable at accounting for the types of mistakes that people actually make

in real life. And the Pathfinder program has performed exceptionally well on the examples. It accurately selects the most likely explanation for the user's query, gives plausible probabilities as a measure of confidence in the answer, and is efficient enough to be incorporated into a full intelligent interface.

7.2 Future Directions

There are several areas where the research presented in this dissertation can be extended.

Violated Optimization

One very bothersome weakness in the detection algorithm is a lack of any way to detect violated optimization. Clearly, the full planner-simulator method suggested on page 87 is unacceptable. People do not appear to use anything that complicated, and yet they are able to detect violated optimization in a large number of cases. The question is, what method do people use, and can it be automated?

One possibility is that people have already "pre-selected" what they believe to be the optimal plan in a given situation. If another person suggests a different plan, they only need to worry about two plans (the pre-selected plan and the new plan), which could be simulated and compared rapidly. This is obviously overly simplistic (for example, what about cases where the expert can't say how optimal the best solution is, or even *what* the best solution is, but can still judge whether the user's plan is "good"?). But it might be somewhere to start.

Temporal Reasoning

The temporal reasoning capabilities that are currently used by Pathfinder are quite weak. A simple partial ordering on the substeps of a plan is not expressive enough to handle many of the real plans that people have. Among other things, there is no way to say that one step must occur *during* another step, or that two steps overlap by some amount of time. It is also difficult to discuss temporal events in the state of the world; there is no way to express the fact that a step will take place ten minutes from now, and it is awkward to state that a particular action did *not* take place. A richer temporal logic [2, 112] is needed to handle these cases.

Misconception Features

The work on the misconception features is far from complete; many of the features have not been fully implemented yet. Although Pathfinder already assigns fairly accurate probabilities to the misconceptions, there are still many situations where it could potentially come up with the wrong answer. For example, since permanence is currently not used at all, a misconception resulting from an out-of-date plan hierarchy would be improperly weighted.

Also, some of the methods that are already implemented for calculating the features are a bit naive. Certain types of similarity are measured in a rather crude fashion (such as matching Schedule Y1 and Schedule Y2 because of their syntactic similarity), and other types are overly simplistic. Some new approaches to similarity, which might be particularly relevant for substituted steps, are being explored by researchers in case-based reasoning to help retrieve similar cases [4, 8, 13, 55, 84, 108].

Multiple Goals and Multiple Users

Pathfinder currently assumes that there is a single user with a single top-level goal. But plan recognition in real life does not always work that way. People generally have many background goals that they are trying to achieve simultaneously [110], which complicates the recognition process and leads to more combinations of misconceptions, due to the interactions of the multiple goals. And there are often multiple agents, who may be collaborating in a single plan [44, 56]. This makes the user models more difficult to manage, since there may be more than one user model active at a given time. These issues are not addressed at all in the current system.

A Full Interface System

Finally, it would be very interesting to incorporate the ideas of this thesis into a full intelligent interface system. Pathfinder is designed to be only one component of such a system, along with a natural language understander, an expert system, and a response generator. There are several questions that cannot be answered by just looking at a single component. For example, there are many knowledge representation and natural language issues that will not arise until Pathfinder is hooked up to a complete system. What difficulties will there be in providing the semantic information that Pathfinder requires? What issues are involved in constructing an appropriate response that can cover

multiple equally likely explanations? There will clearly be other interesting research issues that arise when the full system is put together. But once this is accomplished, it will give intelligent interfaces the ability to accept live English input and progress all the way to an individualized English response that is tailored to address the user's specific misconceptions.

Bibliography

[1] James Allen. *Natural Language Understanding.* Benjamin/Cummings Publishing Co., Menlo Park, California, 1987.

[2] James F. Allen and Patrick J. Hayes. A common-sense theory of time. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 528–531, Los Angeles, California, August 1985.

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

[4] Richard Alterman and Mary Wentworth. Determining the important features of a case. In *Proceedings of the Proceedings of the Case-Based Reasoning Workshop*, pages 193–196, Pensacola Beach, Florida, May 1989.

[5] John R. Anderson. The expert model. In Martha C. Polson and J. Jeffrey Richardson, editors, *Foundations of Intelligent Tutoring Systems*, pages 21–53. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1988.

[6] John R. Anderson, C. Franklin Boyle, and Brian J. Reiser. Intelligent tutoring systems. *Science*, 288:456–462, 1985.

[7] John R. Anderson, C. Franklin Boyle, and Gregg Yost. The geometry tutor. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1–7, Los Angeles, California, August 1985.

[8] Kevin D. Ashley. Assessing similarities among cases: A position paper. In *Proceedings of the Proceedings of the Case-Based Reasoning Workshop*, pages 72–76, Pensacola Beach, Florida, May 1989.

[9] Jerome Azarewicz, Glenn Fala, Ralph Fink, and Christof Heithecker. Plan recognition for airborne tactical decision making. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 805–811, Philadelphia, Pennsylvania, August 1986.

[10] A. Bagchi and A. Mahanti. Search algorithms under different kinds of heuristics — a comparative study. *Journal of the ACM*, 30(1):1–21, 1983.

[11] A. Bagchi and A. Mahanti. Three approaches to heuristic search in networks. *Journal of the ACM*, 32(1):1–27, 1985.

[12] Robert W. Bailey, Stephen T. Demers, and Allen I. Lebowitz. Human reliability in computer-based business information systems. *IEEE Transactions on Reliability*, R-22(3):140–147, 1973.

[13] Ray Bareiss and James A. King. Similarity assessment in case-based reasoning. In *Proceedings of the Proceedings of the Case-Based Reasoning Workshop*, pages 67–71, Pensacola Beach, Florida, 1989.

[14] Carol A. Broverman and W. Bruce Croft. Reasoning about exceptions during plan execution monitoring. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 190–195, Seattle, Washington, August 1987.

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

[16] Richard R. Burton. Diagnosing bugs in a simple procedural skill. In D. Sleeman and J.S. Brown, editors, *Intelligent Tutoring Systems*, pages 157–183. Academic Press, San Diego, California, 1982.

[17] Randall J. Calistri. 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.

[18] Randall J. Calistri. Current research in explanation-based reasoning at GE-CRD. Technical report, GE Research & Development Center, Artificial Intelligence Program, Schenectady, New York, forthcoming 1989.

[19] Randall J. Calistri. A modified A* algorithm for robust plan recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, June 1990.

[20] Sandra Carberry. Tracking user goals in an information-seeking environment. In *Proceedings of the National Conference on Artificial Intelligence*, pages 59–63, Washington, D.C., August 1983.

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

[22] John M. Carroll and Amy P. Aaronson. Learning by doing with simulated help. *Communications of the ACM*, 31(9), September 1988.

[23] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[24] Eugene Charniak. Knowledge representation and the plan-recognition-as-planning^{-1} theory. Technical Report CS-85-01, Department of Computer Science, Brown University, Providence, Rhode Island, 1985.

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

[26] Eugene Charniak and Drew McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley Publishing Co., 1986.

[27] Brant A. Cheikes. The architecture of a cooperative respondent. Technical Report MS-CIS-89-13, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania, 1989.

[28] Philip R. Cohen and C. Raymond Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3:177–212, 1979.

[29] Robin Cohen. Dimensions of user modeling for disambiguating complex input. In *Proceedings of the 1989 IJCAI Workshop on A New Generation of Intelligent Interfaces*, pages 33–36, Detroit, Michigan, August 1989.

[30] Margaret E. Connell, Karen E. Huff, and Victor R. Lesser. Implementing an incremental hierarchical plan recognition system. In *Proceedings of the Second AAAI Workshop on Plan Recognition*, Detroit, Michigan, August 1989.

[31] W.B. Croft and L.S. Lefkowitz. Using a planner to support office work. In *Proceedings of the ACM Conference on Office Information Systems*, pages 55–62, 1988.

[32] Henry W. Davis, Anna Bramanti-Gregor, and Xiaofeng Chen. Torward finding optimal solutions with non-admissible heuristics: A new technique. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 303–308, Detroit, Michigan, August 1989.

[33] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3):505–536, 1985.

[34] William Feller. *An Introduction to Probability and its Applications.* John Wiley and Sons, Ltd., London, England, 1968.

[35] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[36] Gerhard Fischer. Enhancing incremental learning processes with knowledge-based systems. In Heinz Mandl and Alan Lesgold, editors, *Learning Issues for Intelligent Tutoring Systems*, pages 138–163. Springer-Verlag, New York, New York, 1988.

[37] Norman D. Geddes. Intent inferencing using scripts and plans. In *Proceedings of the First Annual Aerospace Applications of Artificial Intelligence Conference*, pages 160–172, 1985.

[38] Michael R. Genesereth. The role of plans in automated consultation. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 311–319, Tokyo, Japan, August 1979.

[39] Michael R. Genesereth. The role of plans in intelligent teaching systems. In D. Sleeman and J.S. Brown, editors, *Intelligent Tutoring Systems*, pages 137–155. Academic Press, San Diego, California, 1982.

[40] Robert Goldman and Eugene Charniak. A probabilistic ATMS for plan recognition. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, Minnesota, August 1988.

[41] Robert Goldman and Eugene Charniak. 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*, pages 1074–1079, Detroit, Michigan, August 1989.

[42] Bradley A. Goodman and Diane J. Litman. Plan recognition for intelligent interfaces. In *Sixth IEEE Conference on Artificial Intelligence Applications*, 1990.

[43] Richard H. Granger, Jr. When expectation fails: Towards a self-correcting inference system. In *Proceedings of the First Annual National Conference on Artificial Intelligence*, pages 301–305, Stanford, California, August 1980.

[44] Barbara Grosz and Candy Sidner. Plans to support communication and collaboration among agents. IJCAI Plan Recognition Workshop (unpublished paper), August 1989.

[45] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107, 1968.

[46] William Lewis Johnson. *Intention-Based Diagnosis of Errors in Novice Programs.* PhD thesis, Department of Comptuer Science, Yale University, New Haven, Connecticut, 1985.

[47] Aravind Joshi, Bonnie Webber, and Ralph M. Weischedel. Default reasoning in interaction (extended abstract). In *Workshop on Non-Monotonic Reasoning*, pages 145–150, 1984.

[48] Aravind Joshi, Bonnie Webber, and Ralph M. Weischedel. Living up to expectations: Computing expert responses. In *Proceedings of the National Conference on Artificial Intelligence*, pages 169–175, Austin, Texas, August 1984.

[49] Aravind Joshi, Bonnie Webber, and Ralph M. Weischedel. Preventing false inferences. In *Proceedings of the 22nd Annual Meeting of the Association for Computational Linguistics*, pages 134–138, Stanford, California, 1984.

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

[51] Robert Kass and Tim Finin. Rules for the implicit acquisition of knowledge about the user. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 295–300, Seattle, Washington, August 1987.

[52] Robert Kass and Tim Finin. Modelling the user in natural language systems. *Computational Linguistics*, 14(3):5–22, September 1988.

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

[54] Henry A. Kautz and James F. Allen. Generalized plan recognition. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 32–37, Philadelphia, Pennsylvania, August 1986.

[55] Janet L. Kolodner. Judging which is the "best" case for a case-based reasoner. In *Proceedings of the Proceedings of the Case-Based Reasoning Workshop*, pages 77–81, Pensacola Beach, Florida, May 1989.

[56] Kurt Konolige and Martha E. Pollack. Ascribing plans to agents: Preliminary report. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 924–930, Detroit, Michigan, August 1989.

[57] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[58] Richard E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88, 1987.

[59] Richard E. Korf. Real-time path planning. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, Austin, Texas, 1987.

[60] Diane J. Litman. Understanding plan ellipsis. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 619–624, Philadelphia, Pennsylvania, August 1986.

[61] Diane J. Litman and James F. Allen. A plan recognition model for clarification subdialogues. In *Proceedings of the 22nd Annual Meeting of the Association for Computational Linguistics*, pages 302–311, Stanford, California, July 1984.

[62] David C. Littman, Jeannine Pinto, and Elliot Soloway. An analysis of tutorial reasoning about programming bugs. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 320–326, Philadelphia, Pennsylvania, August 1986.

[63] Marc Luria. Expressing concern. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 221–227, Stanford, California, July 1987.

[64] Marc Luria. *Knowledge Intensive Planning.* PhD thesis, Computer Science Division (EECS), University of California, Berkeley, California, 1988.

[65] M. Matz. Towards a process model for high school algebra errors. In D. Sleeman and J.S. Brown, editors, *Intelligent Tutoring Systems*, pages 25–50. Academic Press, San Diego, California, 1982.

[66] Eric Mays. Failures in natural language systems: Applications to data base query systems. In *Proceedings of the First Annual National Conference on Artificial Intelligence*, pages 327–330, Stanford, California, August 1980.

[67] Kathleen F. McCoy. The ROMPER system: Responding to object-related misconceptions using perspective. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 97–105, New York, New York, June 1986.

[68] Kathleen F. McCoy. Reasoning on a highlighted user model to respond to misconceptions. *Computational Linguistics*, 14(3):52–63, September 1988.

[69] Kathleen F. McCoy. Generating context sensitive responses to object-related misconceptions. *Artificial Intelligence*, Forthcoming, 1989.

[70] Jean McKendree and Jay Zaback. Planning for advising. In *CHI-88*, pages 179–183, 1988.

[71] Henry B. McLoughlin. Personae: Models of stereotypical behavior. In R.G. Reilly, editor, *Communication Failure in Dialogue and Discourse*, pages 233–241. North-Holland, Amsterdam, Holland, 1987.

[72] William R. Murray. Heuristic and formal methods in automatic program debugging. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 15–19, Los Angeles, California, August 1985.

[73] Leon H. Nawrocki, Michael H. Strub, and Ross M. Cecil. Error categorization and analysis in man-computer communication systems. *IEEE Transactions on Reliability*, R-22(3):135–140, 1973.

[74] Maurine Neiberg and Eugene Charniak. Plan recognition failure in a subtractive model of question answering. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, Minnesota, August 1988.

[75] Nils J. Nilsson. *Principles of Artificial Intelligence.* Morgan Kaufmann Publishers, Inc, Los Altos, California, 1980.

[76] Stellan Ohlsson and Pat Langley. Identifying solution paths in cognitive diagnosis. Technical Report CMU-RI-TR-85-2, The Robotics Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania, March 1985.

[77] Stellan Ohlsson and Pat Langley. Psychological evaluation of path hypotheses in cognitive diagnosis. In Heinz Mandl and Alan Lesgold, editors, *Learning Issues for Intelligent Tutoring Systems*, pages 42–62. Springer-Verlag, New York, New York, 1988.

[78] Andrew J. Palay. *Searching with Probabilities.* Pitman, London, England, 1985.

[79] Cecile L. Paris. Description strategies for naive and expert users. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 238–245, Chicago, Illinois, July 1985.

[80] Cecile L. Paris. Tailoring object descriptions to a user's level of expertise. *Computational Linguistics*, 14(3):64–78, September 1988.

[81] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1984.

[82] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, Inc, Los Altos, California, 1988.

[83] Martha E. Pollack. *Inferring Domain Plans in Question-Answering.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania, 1986.

[84] Bruce W. Porter. Similarity assessment: Computation vs representation. In *Proceedings of the Proceedings of the Case-Based Reasoning Workshop*, pages 82–84, Pensacola Beach, Florida, May 1989.

[85] Alex Quilici, Michael G. Dyer, and Margot Flowers. Recognizing and responding to plan-oriented misconceptions. *Computational Linguistics*, 14(3):38–52, September 1988.

[86] Brian J. Reiser, John R. Anderson, and Robert G. Farrell. Dynamic student modelling in an intelligent tutor for lisp programming. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 8–14, Los Angeles, California, August 1985.

[87] Brian J. Reiser, Daniel Y. Kimberg, Marsha C. Lovett, and Michael Ranney. Knowledge representation and explanation in GIL, an intellgigent tutor for programming. Technical Report CSL-37, Princeton University, 1989.

[88] Brian J. Reiser, Michael Ranney, Marsha C. Lovett, and Daniel Y. Kimberg. Facilitating students' reasoning with causal explanations and visual representations. In *Proceedings of the Fourth International Conference on Artificial Intelligence and Education*, pages 228–235, 1989.

[89] Elaine Rich. User modeling via stereotypes. *Cognitive Science*, 3:329–354, 1979.

[90] William B. Rouse, Norman D. Geddes, and John M. Hammer. Computer-aided fighter pilots. *IEEE Spectrum*, pages 38–41, March 1990.

[91] William B. Rouse and Sandra H. Rouse. Analysis and classification of human error. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(4):539–549, 1983.

[92] Earl Sacerdoti. Planning in a hierarchy of abstraction spaces. In *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pages 412–422, Stanford, California, August 1973.

[93] Earl D. Sacerdoti. The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 206–214, Tbilisi, Georgia, USSR, September 1975.

[94] 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.

[95] Charles F. Schmidt and John D'Addamio. A model of the common-sense theory of intention and personal causation. In *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pages 465–471, Stanford, California, August 1973.

[96] Ethel Schuster. Grammars as user models. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 20–22, Los Angeles, California, August 1985.

[97] Marc M. Sebrechts and Lael J. Schooler. Diagnosing errors in statistical problem-solving: Associative problem recognition and plan-based error detection. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pages 691–703, Seattle, Washington, July 1987.

[98] Anup K. Sen and A. Bagchi. Fast recursive formulations for best-first search that allow controlled use of memory. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 297–302, Detroit, Michigan, August 1989.

[99] Candace L. Sidner and David J. Israel. Recognizing intended meaning and speakers' plans. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 203–208, Vancouver, B.C., Canada, August 1981.

[100] W.T. Singleton. Theoretical approaches to human error. *Ergonomics*, 16(6):727–737, 1973.

[101] James R. Slagle and Philip Bursky. Experiments with a multipurpose, theorem-proving heuristic program. *Journal of the ACM*, 15(1):85–99, 1968.

[102] D. Sleeman. Assessing aspects of competence in basic algebra. In D. Sleeman and J.S. Brown, editors, *Intelligent Tutoring Systems*, pages 185–199. Academic Press, San Diego, California, 1982.

[103] D. Sleeman. Some challenges for intelligent tutoring systems (extended abstract). In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1166–1168, Milan, Italy, August 1987.

[104] D. Sleeman and J.S. Brown. Introduction: Intelligent tutoring systems. In D. Sleeman and J.S. Brown, editors, *Intelligent Tutoring Systems*, pages 1–11. Academic Press, San Diego, California, 1982.

[105] James C. Spohrer, Elliot Soloway, and Edgar Pope. A goal/plan analysis of buggy pascal programs. Technical Report YALEU/CSD/RR#392, Department of Comptuer Science, Yale University, New Haven, Connecticut, 1985.

[106] Albert Stevens, Allan Collins, and Sarah E. Goldin. Misconceptions in students' understanding. In D. Sleeman and J.S. Brown, editors, *Intelligent Tutoring Systems*, pages 13–24. Academic Press, San Diego, California, 1982.

[107] Russell H. Taylor, Matthew T. Mason, and Kenneth Y. Goldberg. Sensor-based manipulation planning as a game with nature. In *Proceedings of the Fourth International Symposium on Robotics Research*, August 1987.

[108] Paul Thagard and Keith J. Holyoak. How to computer semantic similarity. In *Proceedings of the Proceedings of the Case-Based Reasoning Workshop*, pages 85–86, Pensacola Beach, Florida, 1989.

[109] Amos Tversky. Features of similarity. *Psychological Review*, 84:327–352, 1977.

[110] Peter van Beek. A model for generating better explanations. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 215–220, Stanford, California, July 1987.

[111] Joost M. van Eekhout and William B. Rouse. Human errors in detection, diagnosis, and compensation for failures in the engine control room of a supertanker. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(12):813–816, 1981.

[112] Marc Vilain and Henry Kautz. Constraint propogation algorithms for temporal reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 377–382, Philadelphia, Pennsylvania, August 1986.

[113] Bonnie Lynn Webber and Eric Mays. Varieties of user misconceptions: Detection and correction. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 650–652, Karlsruhe, West Germany, August 1983.

[114] Robert Wilensky. Why John married Mary: Understanding stories involving recurring goals. *Cognitive Science*, 2(3):235–266, 1978.

[115] Robert Wilensky. Talking to UNIX in English: An overview of an on-line consultant. Technical Report UCB/CSD82/104, Computer Science Division (EECS), University of California, Berkeley, California, 1982.

[116] Robert Wilensky, David N. Chin, Marc Luria, James Martin, James Mayfield, and Dekai Wu. The Berkeley UNIX consultant project. *Computational Linguistics*, 14(4):35–84, December 1988.

[117] David E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann Publishers, Inc, Los Altos, California, 1988.