Controlling inference in plan recognition

JAMES MAYFIELD
Computer Science Department
University of Maryland at Baltimore County
Baltimore, MD 21228-5398, USA
mayfield@umbc3.umbc.edu

(Submitted 13 September 1990; final version accepted May 30, 1991)

Abstract. An algorithm based on an assessment of the completeness of an explanation can be used to control inference in a plan recognition system: If the explanation is complete, inference is stopped. If the explanation is incomplete, inference is continued. If it cannot be determined whether the explanation is complete, then the system weighs the strength of its interest in continuing the analysis against the estimated cost of doing so. This algorithm places existing heuristic approaches to the control of inference in plan recognition into a unified framework. The algorithm rests on the principle that the decision to continue processing should be based primarily on the explanation chain itself, not on external factors. Only when an analysis of the explanation chain proves inconclusive should outside factors weigh heavily in the decision. Furthermore, a decision to discontinue chaining should never be final; other components of the system should have the opportunity to request that an explanation chain be extended. An implementation of the algorithm, called PAGAN, demonstrates the usefulness of this approach.

Key words: plan recognition, abduction, control of inference, explanation, natural language understanding, consultation systems.

1. Introduction

Plan recognition is typically performed by chaining together a sequence of abductive inferences. Each inference is made by using a plan schema, which associates a plan with a goal, to explain an action. To continue the analysis of a particular action is to extend the chain of inferences about that action by inferring an additional plan schema to explain the most recently inferred goal. I call such a chain of inferences an *explanation chain.*

As an example, consider the utterance:

USER: Can you tell me how to edit a file?[1]

One possible explanation chain of this question contains the following goals:

[1] The examples in this paper are drawn from the UNIX consulting domain, with the assumption that the user asking questions of the consultant is a novice. UC, a dialogue system that acts as a consultant for the UNIX operating system, is described in the implementation section. UNIX is a trademark of AT&T Bell Laboratories.

1. The user wants to request that you tell her how to edit a file
2. The user wants to know how to edit a file
3. The user wants to edit a file
4. The user wants to insert the text of a paper into a file
5. The user wants to print a paper
6. The user wants to publish a paper
7. The user wants to get tenure

Each of these goals represents the result of one iteration of the plan recognition algorithm. The first goal directly explains the utterance. Each of the subsequent goals provides an explanation for the preceding goal, and therefore provides part of an explanation for the utterance. Thus, an explanation of an utterance comprises an *entire* explanation chain.[2]

A system that is designed to be general enough to handle a wide range of inputs will represent the knowledge needed to make some or all of the above inferences separately. However, not all of these inferred goals will be relevant to every system that must handle the utterance in question. Consequently, the plan recognizer must determine which of these inferences it should make, and which it should avoid making. Since each inference is built on the preceding inference, this decision amounts to determining when the chaining process should continue, and when it should stop.

Notice that these inferences are all abductive inferences. An abductive inference combines a rule of the form 'A implies B' with an observation of B to conclude the possible existence of A. For example, the above analysis combines the rule 'one way to find out something is to ask someone' with an observation of a particular asking (goal 1) to infer that the user wants to know something (goal 2). In general, plan recognition will require a mix of abductive and deductive inferences. For example, consider the exchange:

UC: When do you need the file restored?
USER: The paper is due Friday.

In this example, abductive inference alone can determine that the user wants the system to know that the paper is due on Friday. However, deductive inference is required to infer the relationship between the time at which the file is restored and the time at which the paper is submitted. In this paper I am concerned only with the control of abductive inference; I will use the term 'inference' to mean abductive inference.

Another topic that I do not address in this paper is ambiguity. When more than one plan schema is found that might explain an action, an ambiguity is

[2] Many systems require the plan recognizer to select a single goal from an explanation chain to be addressed by the system. However, the planning and agency components of the system are in a better position to make such a selection; therefore, in the proposed model the plan recognizer returns the chain of *all* inferred goals.

introduced. Unless the ambiguity can be immediately resolved, two or more competing explanation chains will need to be developed. However, since the detection and resolution of ambiguity is orthogonal to the development of each individual chain of goals, I do not dwell on these issues here.

My purpose in this paper is fourfold. First, I discuss previous approaches to the problem of determining when to continue plan recognition, and show how certain of these approaches are flawed. Second, I introduce an algorithm for determining when plan recognition should continue, and show an implementation of that algorithm. Third, I demonstrate how an outside process can request the continuation of an analysis that the algorithm has previously discontinued. Finally, I give an analysis of the need for control of inference, and an analysis of the performance of the proposed chaining algorithm.

2. Previous approaches

Some systems make every possible inference. This is an acceptable approach because their domains are highly constrained. In a less constrained environment in which the system must deal with a wide range of human affairs, this approach is not tenable. This is true even if the particular topic of discussion is quite limited, because the system will still need access to a wide range of world knowledge. Therefore, some method for determining when to cut off the inference process is needed.

2.1. Rule-based methods

The following rules have been proposed for deciding when to halt chaining in a plan recognition system:

Rule 1. Halt when the plan inferred serves a known goal (Mann et al. (MML77))

Rule 2. Halt when a thematic explanation is found (Wilensky (Wil78))

Rule 3. Halt when the goal is uninteresting (Schank (Sch79))

Rule 4. Halt when a meta-plan to a known plan is found (Litman (Lit85))

Rule 5. Halt when a domain goal is encountered (Litman (Lit85))

Rule 6. Halt when an ambiguity is encountered (Sidner (Sid85))

Rule 1 says that chaining should stop when an inferred explanation matches some expected goal. Rule 2 introduces themes to serve as high-level expectations, thereby allowing Rule 1 to be applicable when there has been no preceding dialogue. Rule 3 proposes that the decision about when to chain should be based not only on the content of the utterance, but also on the hearer's attitudes towards the utterance and the inferred goals. Rule 4 expands the notion of how an utterance can connect to previous dialogue by viewing it as a meta-plan to some existing dialogue plan. Rule 5 supports the position that chaining should stop as soon as some goal in the domain of discourse is inferred. Finally, Rule 6 says that chaining should stop when two or more explanations of the same event are found that cannot be disambiguated immediately.

Each of rules 1-4 will be incorporated into the chaining control algorithm in the following section. Rules 5 and 6 concern the termination of chaining when the explanation cannot be grounded in knowledge of the preceding dialogue. Each of these rules is problematic. Rule 5 is problematic because the first domain goal encountered might not be the important one. Consider the question:

USER: Can you tell me how I can compact my directory?

The first domain goal that is encountered in the analysis of this utterance is the goal of compacting a directory. If analysis is halted once this goal is inferred, there is no way that the system can suggest for example that the user request a larger disk quota from the system manager. To do so would require that the system know of the user's goal of gaining more disk space. If chaining is stopped once the goal of compacting the directory has been inferred, this motivating domain goal will never be discovered.

Rule 6 is also problematic. Sidner's reason for discontinuing processing when an ambiguity is detected is that a participant in a dialogue cannot expect a hearer to react to an utterance based on "the assumption that somehow the hearer will guess at the information that is needed for the hearer to make an intended response" [p. 4]. This is true; however, it does not follow that processing should be halted when an ambiguity is encountered. There are two reasons that it might be important to continue analysis after an ambiguity has been discovered. First, such subsequent processing might itself prove important for the analysis of the utterance. The best example of this is when further processing allows the ambiguity to be resolved. For example, consider the sequence:

USER: I want more disk space.
USER: Does mv delete files?

This last question is ambiguous between asking what mv does and asking how to delete files. Knowing only this, the ambiguity cannot be resolved. However, if the ambiguity is temporarily ignored and both interpretations are pursued, the latter interpretation will ultimately be selected. This is because deleting files is a way to increase disk space, and therefore matches a goal introduced by the preceding statement.

The second reason that it might be important to continue processing after an ambiguity is detected is that the speaker might not be aware that the ambiguity is present. A speaker never has a perfect model of the hearer. Consequently, an utterance that is believed by its speaker to be unambiguous might be perceived by a hearer as ambiguous. In such cases, it is important for the hearer to fully develop the competing interpretations, even if the ambiguity cannot be resolved. Doing so gives the components of the system outside of the plan recognizer the greatest latitude in deciding which goals to address. See Appelt and Pollack (AP91), Calistri-Yeh (CY91), Eller and Carberry (EC91), Pollack (Pol86), Retz-Schmidt (RS91), and Quilici *et al.* (QDF88) for examples of systems that deal with mismatches between the beliefs of speakers and hearers.

2.2. Resource-based and ratings-based methods

The approach to chaining implicit in the above rules bases the decision to chain on each competing explanation chain individually. That is, when a goal is inferred, the decision about whether that goal must itself be explained does not depend on what other potential explanation chains are extant. In contrast to this approach is the resource-based method, exemplified by Allen (All79) and Appelt and Pollack (AP91). In this method, decisions are made not about whether an explanation for a goal is needed, but rather about where processing power should be directed. Allen's system for example had a set of heuristics for rating the strength of an explanation. At each iteration, processing was done only on the single explanation with the highest rating. Thus the decision about whether to chain on a particular explanation was based not only on the attributes of that explanation, but also on attributes of all other explanations under consideration.

An advantage of the resource-based approach to chaining is that it provides a fast and effective means for disambiguation. However, there are two main problems with this method. First, it fails when a single utterance is made in service of multiple goals. In such cases, this approach will find only one chain of goals that motivated the utterance. Appelt (App85) and Norvig (Nor88) describe situations in which a single utterance is used in service of more than one goal. Secondly, this approach assumes that the processor is a scarce resource. With the development of parallel architectures and languages, it is becoming increasingly feasible to carry forward multiple lines of reasoning. A plan recognizer should be able to take advantage of such parallelism to develop multiple competing explanations.

It is possible to use a resource-based scheme such as Allen's to compare competing explanations against a threshold rather than against each other. Doing so converts the resource-based approach into a ratings-based approach. For example, Rieger (Rie75) estimates the *likelihood* that an explanation is true, and the *significance* of the explanation; if the product of these two factors drops below a particular threshold, inference on that explanation is terminated.

The ratings-based approach suffers from the problem that it is difficult to determine which factors cause chaining to terminate. It is consequently cumbersome to properly adjust the rating scheme by hand. One way to correct this problem is to train a system that uses this approach on previously-analyzed test data. See Soh (Soh90) for an example of the use of training in a semantic network for abductive reasoning.

The rules described above represent pieces of the solution to the chaining problem. What has been lacking in the literature is not heuristics to apply to the chaining problem so much as a framework that ties together the available heuristics. In the following section I propose such a framework. I then show how the valid heuristics itemized above (rules 1–4), as well as some new heuristics, can be placed into this framework to form a complete approach to the chaining problem.

3. Algorithm

The purpose of a chaining control mechanism is to balance two competing aims: the creation of a *complete* explanation,[3] and the conservation of computing resources. There are two kinds of completeness of an explanation:

1. Depth completeness
2. Breadth completeness

Depth completeness is completeness of a single line of explanation of an utterance. An explanation of an utterance exhibiting depth completeness includes a goal that motivated the production of the utterance, and a goal or theme to explain each inferred goal. *Breadth completeness* is coverage of all aspects of the utterance. An explanation that exhibits breadth completeness includes every goal that motivated the production of the utterance. A chaining control mechanism then is concerned with balancing the need for depth completeness with the need to conserve computing resources; hereafter, I will use the term 'completeness' to mean depth completeness.

The framework for deciding when to chain that I propose is based on two assessments:

1. An assessment of the completeness of the explanation chain
2. An assessment of the practicability of chaining

The decision about whether to chain is dominated (but not dictated) by whether the plan recognizer believes the explanation chain to be complete. If the explanation chain is complete, there is no reason to continue trying

[3] Completeness is one criterion for evaluating the quality of an explanation. A comprehensive set of such criteria is given in Mayfield (Mayar).

to explain the utterance. If the explanation chain is not complete, then under ideal circumstances processing should continue until it is complete. Thus, the first step in deciding whether to chain is to determine whether the explanation chain in question is complete.

Of course, it is not always possible to determine whether an explanation chain is complete. Thus, the function that examines an explanation chain to determine whether it is complete must indicate that one of three conditions holds:

1. The explanation chain is complete
2. The explanation chain is incomplete
3. It cannot be determined whether the explanation chain is complete

Once the determination has been made as to whether an explanation chain is complete, that knowledge must be used to decide whether to continue processing. Ideally, processing would stop if the explanation chain were complete, and would continue otherwise. This would always result in a complete explanation, if the system had adequate information available to it. However, the difficulty of determining whether an explanation chain is complete, together with resource limitations, prevent this strategy from being a practical one. Consequently, if it can be determined that the explanation chain is certainly complete, or that the explanation chain is certainly incomplete, the decision about whether to chain should be based on this determination. However, if it cannot be determined whether the explanation chain is complete, the plan recognizer must decide whether to chain by assessing the benefits of chaining and comparing these benefits against the cost of chaining. If the benefits outweigh the cost, chaining continues; if the converse holds, chaining halts. This limits computational expense, at the cost of producing an explanation that is incomplete. Smith (Smi89) describes a similar cost/benefits analysis for the control of backward chaining in a deductive retrieval system.

The completeness of an explanation chain is largely based on the characteristics of that explanation chain. In contrast, the cost/benefit analysis of chaining is primarily based on the status of the plan recognizer's processing. Neither the cost assessment nor the benefits assessment can be made directly without actually carrying out the analysis. Therefore, the cost of chaining is determined by estimating the amount of work that will be required to find all reasonable explanations for the goal in question. The benefits of chaining are estimated by assessing the plan recognizer's *curiosity* about the goal. Both of these estimates rely not only on knowledge of the goal to be explained, but also on knowledge of the system's goals and capabilities.

When the plan recognizer cannot determine whether an explanation chain is complete or incomplete, it estimates the costs and the benefits of chaining. These estimates are used in the following way. If the strength of one exceeds the strength of the other by a threshold, the stronger one is given precedence. For example, if the system's curiosity about the explanation is high, and the cost of continuing the analysis appears to be low, chaining continues. If on the other hand the two are of roughly equal strength, then analysis is discontinued.

In the following subsections, I first show how an explanation chain can be deemed to be complete or incomplete. I then discuss how a cost/benefits analysis of chaining can be performed.

3.1. Completeness of an explanation chain

There are two conditions that indicate to the plan recognizer that an explanation chain is complete:

1. The explanation chain has previously been determined to be complete
2. The explanation chain meets established criteria for completeness

First, the explanation chain might already have been determined to be complete. This is the case when an explanation chain matches an expectation. Expectation matching is a way to connect an utterance to a representation of the preceding dialogue, thereby grounding the utterance in the dialogue. Since the plan recognizer already has an adequate explanation for any action that it expects, no further work is required to satisfy its needs in this situation. Note that if metaplans are viewed as plausible steps of existing plans, they can be detected during expectation matching.

The second way that the plan recognizer can detect completeness of an explanation chain is by applying established completeness criteria. Such criteria are derived from knowledge of the system's own domain of expertise. This use of knowledge of the domain of expertise, that is, knowledge of the system's contract with the user, stems directly from the principle that an explanation should be applicable to the needs of the system that is using it (Mayar). An explanation chain is declared to be complete when the most recently inferred goal is no longer applicable to the purposes of the system. Consider for example the following question:

User: How can I read a phone number that's in Beth's directory?

An explanation of this utterance might contain the goal of making a phone call. Such an explanation chain is complete for UC (whether or not it meets other criteria for good explanations) because the goal of making a phone call is not within the UC domain.

Themes (in the sense of Schank and Abelson (SA77)) can be used to indicate completeness. In this example, the existence of a 'using the phone' theme might indicate to the system that using the phone is something that people do that does not require explanation. However, since the existence of a thematic explanation of an action does not automatically preclude a relevant intentional explanation of that action, the system designer must be careful to include only themes that lie clearly outside the domain of the system.

3.2. Incompleteness of an explanation chain

In addition to determining that an explanation chain is complete, it might also be possible to determine that it is incomplete. There are two approaches to judging the incompleteness of an explanation chain:

1. Examination of the character of the explanation itself
2. Examination of expectations derived from the preceding dialogue

Some goals seem to fall naturally under the rubric of incomplete explanations. Such goals are called *instrumental goals*. According to Schank and Abelson (SA77), an instrumental goal is one that arises only to serve another goal. Schank and Abelson's concept of an instrumental goal is an absolute one; a particular goal is either an instrumental goal or it isn't. Thus, getting a loan is always instrumental to using the money. In contrast to this position, I claim that what is instrumental is relative to the task at hand. Editing a file is instrumental to writing a dissertation, but at the level at which UC works, it is a top-level goal.

The second way that an explanation chain can be judged to be incomplete is based on knowledge of the preceding dialogue. When there is a strong expectation of a particular action, and the explanation chain built so far neither matches that expectation nor is grounded in the previous dialogue in any way, incompleteness is indicated. This provides a strong impetus to treat the next utterance as a means of achieving the expected action. Consider the exchange:

UC: Is the file in your directory?
User: The file is in /tmp.

Here there is a strong expectation that the user's utterance will somehow relate to UC's question. Consequently, chaining should continue on this utterance until it is determined that the user's utterance constitutes a new plan for UC's goal of learning the location of the file.

3.3. The cost of chaining

It is sometimes expedient to save processing time at the expense of producing a poorer explanation of an utterance. The principle of *difficulty of analysis* holds that analysis of an utterance should be terminated if the cost of continuing the analysis becomes too high. Consider for example the question:

USER: How can I edit a file?

In the absence of other knowledge, the goal of editing a file is a difficult one to explain. There are many reasons that one might want to edit a file. It would be prohibitively expensive to carry out an analysis of each of these explanation chains without some method for constraining the search. The principle of difficulty of analysis provides an impetus for the plan recognizer to terminate processing in such situations.

An alternative to discontinuing processing when the cost of chaining becomes too great is to question the user about his or her intentions. Wu (Wu91) describes a system that generates 'active acquisition goals' when faced with this situation. If the system elects to pursue an active acquisition goal, it will ask the user to explain how the inference chain in question should be completed.

3.4. THE BENEFITS OF CHAINING

The benefits of extending an explanation chain that is clearly incomplete are obvious when one's goal is to produce complete explanations. Chaining is always done on such explanation chains. When the explanation chain in question is not clearly incomplete though, the plan recognizer must base the chaining decision not on the absolute characteristics of the explanation, but rather with how those characteristics relate to the goals of the system. The *principle of curiosity* defines this relationship. This principle directs the plan recognizer to continue analysis on goals about which it is curious. I define *interestingness* as the quality that causes the plan recognizer to be curious about a goal (cf. Schank (Sch79)).

A goal is deemed to be interesting when it is in an interesting goal relationship with another goal. Wilensky (Wil83) identifies four kinds of goal relationships: goal overlap, goal concord, goal conflict, and goal competition. Any of these types of goal relationships can give rise to curiosity based on interestingness. I will focus here on goal competition (which is a negative relationship between the goals of two different agents).

Goal competition is defined by Wilensky as a situation in which "the fulfillment of one [agent's] goals precludes the fulfillment of another's" [p. 58]. When a speaker's goal comes into competition with a goal held by the system, the plan recognizer should be curious about that goal. For example, if a novice user asks:

USER: How do I change the cat[4] program?

the plan recognizer might infer that the user wants to change the cat program. This goal competes with UC's goal of preserving system programs. Because of this competition, the plan recognizer will be curious about why the user would want to change the cat program. It will continue its analysis, possibly arriving at the motivating goal of having a program that is similar in effect to the cat program. This allows UC to propose the 'copy, then change' plan:

UC: First copy the source code with cp, then edit and compile the copy.

The importance of curiosity can be seen by examining the processing of the question:

USER: How could I change a.out[5]?

asked by a novice user. In this example, there are many ways to change a.out. To select the correct one, the planner needs to know what goal motivates the speaker to want to change a.out. This motivating goal is most likely to be the alteration of the behavior of the program. However, if the planner isn't aware of that goal, it might well choose a different plan, such as:

UC: Use emacs.

Using an editor on executable code, while it can theoretically be used to effect a desired programmatic change, is in most cases a ridiculous plan for doing so. An appropriate response would be:

UC: Edit the source code, then recompile it.

The point here is that inferring a higher-level goal can allow the further specification of a lower-level goal. In this example, inferring the goal of changing the program's behavior allows a further specification to be made from 'changing a.out' to 'changing the behavior of a.out.' Since the planner might have more specific plans for more specific goals, the detection of a more specific goal can reveal a better plan.

The above two queries are admittedly extreme examples of a widespread phenomenon in which a speaker selects a poor plan for a reasonable goal. I have included them because it is so clear that chaining must be done on the initial interpretation if the system is to make a reasonable response. A less extreme example of this phenomenon is when a user pursues the goal of reducing disk usage by asking:

[4] Cat is a system program in UNIX that displays the contents of a file.
[5] A.out is a common name for an executable program in UNIX.

USER: How do I save a file to tape?

If chaining continues on this query until the goal of reducing disk space is inferred, the system will have the option of suggesting that the user compress the file and leave it on the disk. In some cases, this plan will be a better one than the user's original plan. The worse the user's original plan, the more important it is to continue chaining until the motivating goal is found, and the more likely it is that the lower-level goal will conflict with some other goal. This is the motivation for the curiosity heuristic.

4. Implementation

The work presented here was done in conjunction with the UNIX Consultant (UC) project (WAC84; WCL+88) at the University of California at Berkeley. UC serves as a consultant for novice users of the UNIX operating system. UC's plan recognition component is called PAGAN (for Plan And Goal ANalyzer) (May89). Once PAGAN has determined the user's goals, the KIP planner (Lur88) is called to try to solve the user's domain goals. The entire system is controlled by the UCEGO mechanism (Chi88), which determines which goals the system should pursue. UC's knowledge is represented in KODIAK (Wil84), a semantic net representation language.

In the following subsections, I discuss some of the implementation details of the concepts discussed in this paper. First, I cover how PAGAN can determine whether a goal is instrumental, and whether it is applicable to the needs of the system. Next, I discuss the identification of a strong expectation for a particular action. Finally, I discuss the two components of determining the practicability of chaining: difficulty of analysis and curiosity. The reader should bear in mind that the methods described in the following subsections are implementation details; other methods for making the required determinations are certainly possible.

4.1. DETERMINING INSTRUMENTALITY AND APPLICABILITY

The most general approach to determining whether a goal is within the system's domain of expertise would take into consideration not only the goal itself and the purposes of the system, but also knowledge of the user and of the preceding dialogue. In practice though, it is more efficient, albeit less flexible, simply to mark explicitly each type of goal that is within the system's domain of expertise. This is the technique that PAGAN uses to determine which goals are in UC's domain of discourse. While this technique cannot change its evaluations as the topic of conversation shifts, it can be a good approximation when the purpose of the system is limited (as it is in the UNIX Consultant). The implementor of such a system must take care to distinguish between goals that are beyond the range of goals that concern the system, and those that are instrumental to goals that concern the system. The latter should be marked not as beyond the system's domain of expertise, but rather as instrumental goals.

As in determining whether a goal is within the system's domain of expertise, determining instrumentality would ideally consider all the circumstances of the utterance, including knowledge of the speaker and of the previous dialogue. However, because of UC's limited domain, PAGAN can use its knowledge of the consulting process to make some assumptions about which goals are instrumental. Thus for example, linguistic goals are instrumental goals, because they are pursued only in order to achieve domain goals. On a more abstract level, any communicative goal is an instrumental goal for PAGAN. Consider the complaint:

USER: I tried typing 'rm foo', but it told me that foo wasn't found.

In this example, the goal of typing is aimed at communicating with UNIX, and is therefore an instrumental goal. A goal of executing a program is also an instrumental goal. In this example, the goal of executing the rm command, which can be inferred from the goal of typing 'rm,' is therefore also an instrumental goal.

Once the goal types contained in the system have been categorized according to instrumentality, these goals can be marked as such in long-term memory. Recognition of instrumental goals is then simply a matter of determining whether the goal under consideration bears the instrumental attribute. Thus, each goal that is included in PAGAN's knowledge base is described there either as an instrumental goal, a domain goal, or a goal that is beyond UC's domain. See the following subsection for a processing example that illustrates these points.

4.2. Identifying strong expectations

Strong expectations are identified by examining the dialogue representation. A strong expectation is indicated when a pending step of a plan is an action to be taken by the user. The strength of the expectation is a function of the recency of the immediately preceding plan step. If the preceding step has just been performed, the strength of the expectation is high. For example, immediately after the system asks a question of the user, there is a strong expectation that the user will respond to the question. On the other hand, if a large number of utterances have occurred since the preceding plan step was executed, the strength of the expectation is low. This corresponds to the intuition that the longer an expectation remains unmet, the less likely it is to be met eventually.

Processing Example

In this and subsequent sections, I present traces of the operation of PAGAN. The following conventions are used in the traces: KODIAK objects are written in capital letters. The sentence that PAGAN is processing is shown at the beginning of each trace, and is written in capital letters surrounded by parentheses within the body of the trace. Descriptions produced by PAGAN as it runs are written in roman, and comments added by hand after-the-fact are written in *italics*. Names that end with a number represent subconcepts of the concept represented by the root name. For example, LOCATION-OF8 is a specialization of the LOCATION-OF concept. Other than deletion of extraneous trace information after-the-fact, and the introduction of comments in italics, the only changes made to PAGAN's trace output by hand are changes in appearance, such as the introduction of font information and the addition of white space.

In this and subsequent examples, only those portions of the trace relevant to PAGAN's decisions about chaining are shown. Furthermore, the set of plan schemas available to PAGAN has been restricted so that no ambiguity is introduced. For a description of how PAGAN handles ambiguity, see Mayfield (May89).

The following trace shows PAGAN's processing of the user's utterance in the exchange:

UC: Is the file in your directory?
User: The file is in /tmp.

In this example, the chaining decision is made three times. The first time, analysis is continued because the inferred goal (that of telling UC that the file is in /tmp) is an instrumental goal, and the explanation chain is therefore incomplete. PAGAN then infers the goal that the hearer of the utterance (i.e. UC) know that the file is in /tmp. The second time the decision is made, analysis is also continued, in this case because there is a strong expectation for a response to the question. The last time, analysis is halted because the user's utterance has been grounded in the preceding dialogue, and the explanation is therefore complete.

UC: Is the file in your directory?
User: The file is in /tmp.
Interpretation produced by the parser: TELL-ACTION36

The first chaining decision is based on the representation of the user's utterance. Since telling is considered to be an instrumental goal, chaining continues:

Determining whether to chain on TELL-ACTION36.
  Determining whether TELL-ACTION36 is a complete explanation of
   (THE FILE IS IN /TMP).
    Determining whether TELL-ACTION36 is outside of UC's domain of
     expertise.
    TELL-ACTION36 is not outside of UC's domain of expertise.
  The explanation is not necessarily complete.

  Determining whether TELL-ACTION36 is an incomplete explanation of
   (THE FILE IS IN /TMP).
    Determining whether TELL-ACTION36 is an instrumental goal.
    TELL-ACTION36 is an instrumental goal.
  The explanation is incomplete.
Chaining should be done on TELL-ACTION36.

Determining whether TELL-ACTION36 is grounded in the preceding dialogue.
TELL-ACTION36 was not grounded in the preceding dialogue.

Found PLANFOR3 as an explanation for TELL-ACTION36.
Synopsis:
 Goal: hearer know whether X;
 Step 1: Tell hearer that X.
Inferred goal is KNOWING-WHETHER14.

The second chaining decision is based on the user's goal of UC *knowing whether the file is in the directory. To demonstrate the use of strong expectations in chaining, I have removed the concept of* KNOWING *(from which* KNOWING-WHETHER14 *is descended) from the set of instrumental goals for this example. However, chaining is still performed, because* PAGAN *has a strong expectation that the user will respond to* UC*'s question (which is represented by* PLAN88*):*

Determining whether to chain on KNOWING-WHETHER14.
  Determining whether KNOWING-WHETHER14 is a complete explanation
   of TELL-ACTION36.
    Determining whether KNOWING-WHETHER14 is outside of UC's
     domain of expertise.
    KNOWING-WHETHER14 is not outside of UC's domain of expertise.
  The explanation is not necessarily complete.

  Determining whether KNOWING-WHETHER14 is an incomplete
   explanation of TELL-ACTION36.
    Determining whether KNOWING-WHETHER14 is an instrumental goal.
    KNOWING-WHETHER14 is not an instrumental goal.
    Determining whether there is a strong expectation for a particular action.
    There is a strong expectation for PLAN88
  The explanation is incomplete.
Chaining should be done on KNOWING-WHETHER14.

Now, PAGAN *tries to determine whether* KNOWING-WHETHER14 *fits into the previous dialogue. It does, so the explanation is complete. Therefore, processing is halted:*

```
Determining whether KNOWING-WHETHER14 is grounded in the preceding
  dialogue.
KNOWING-WHETHER14 is a new plan for the existing goal of
  KNOWING-WHETHER13
Existing plan PLAN87 has been rejected; new plan is PLAN89.
```

4.3. Determining difficulty of analysis

Part of the task of determining difficulty of analysis is subsumed by the task of determining whether a goal is an instrumental goal, a domain goal, or a goal outside of the system's domain. PAGAN assumes that instrumental goals are easy to explain, but that goals outside of the system's domain are quite difficult to explain (since it is unlikely that the system contains the requisite knowledge). Thus, the only goals for which difficulty of analysis must be assessed are domain goals.

Determining the difficulty of analysis for domain goals requires a method to estimate resource usage for continuation of a particular explanation chain. The only way to know this cost for sure is to carry out the analysis. Consequently, a heuristic approach to the problem is called for. One of the simplest such heuristics is to estimate the cost of continuing an explanation chain in proportion to the cost incurred so far by that explanation chain. This can be done by monitoring the total branching that has taken place along the explanation chain. Once this amount exceeds a threshold, any further branching brings the principle of difficulty of analysis into play. The strength of the principle is proportional to the total amount of branching found among all active explanation chains. This is the approach taken by PAGAN.

In practice, this heuristic estimates the cost of continuing an explanation chain well enough to provide PAGAN with significant computational savings (as described below in Section 6.2). However, it is not clear that this heuristic will provide accurate estimates in the general case. In its favor, the heuristic tends to limit the total amount of processing power dedicated to each utterance. The interested reader is referred to Allen's expectation-based and search-based heuristics (All79) as a starting point for the development of an improved heuristic.

4.4. Determining curiosity

The interestingness of a goal is dictated by the relationships that exist between that goal and other goals held by the system and by the user. Wilensky

(Wil78) describes how goal relationships can be detected in the context of story understanding. Often, it is possible to short-cut the process of detecting goal relationships by marking certain concepts as inherently interesting. For example, each of the following questions addresses a goal that might be inherently interesting to UC:

1. USER: How can I get the sources to Nethack?
2. USER: How can I lay waste to a directory?
3. USER: How do I change a.out?

The first example concerns an interesting object: the game of Nethack. The second addresses an interesting action: wreaking havoc. Each of these goals might well be considered inherently interesting by a UNIX Consultant system. The third example addresses the goal of manipulating the contents of a binary file. Neither of the concepts in this example, 'changing a file' and 'binary file,' is interesting in itself. Rather, it is the combination of the two that produces a concept that is interesting. Note that the decision about whether a goal is inherently interesting is not a decision that can be made by the plan recognizer without considering the task of the system in which it is embedded; rather, the criteria for what deserves automatic attention will change as the domain in which the plan recognizer operates changes.

PAGAN uses a hybrid approach to determining curiosity. Each type of goal known to the system has an associated interest level, which is a static integer value. In addition, PAGAN checks for competition between the inferred goal and the system's goals dynamically. The level of curiosity about a goal is the sum of these two values. Other types of goal relationships are ignored by the current implementation of PAGAN.

PROCESSING EXAMPLE

The following trace shows the chaining decisions made by PAGAN in processing the utterance:

USER: How could I change a.out?

The trace illustrates PAGAN's assessment of difficulty of analysis and of curiosity. PAGAN is curious about the goal of changing a binary file. PAGAN therefore chains on the user's goal of changing the binary file, inferring the goal of changing the effect of the program. This and subsequent traces have been condensed to show only those portions relevant to the example at hand. The chain of inferred goals in this example is:

1.	TELL-ACTION38:	Direct interpretation of utterance
2.	KNOWING-THAT15:	User know how to change a.out
3.	CHANGE-EVENT3:	User change a.out
4.	CHANGE-PROGRAM-EFFECT0:	User change effect of a.out

USER: How could I change a.out?
Interpretation produced by the parser: TELL-ACTION38

The first three goals inferred by PAGAN *are instrumental goals, so chaining continues from each of them:*

Determining whether to chain on TELL-ACTION38.
    The explanation is not necessarily complete.
    The explanation is incomplete.
Chaining should be done on TELL-ACTION38.

Found PLANFOR4 as an explanation for TELL-ACTION38.
Synopsis:
  Goal: know something;
  Step 1: ask;
  Step 2: be told.
Inferred goal is KNOWING-THAT15.

Determining whether to chain on KNOWING-THAT15.
    The explanation is not necessarily complete.
    The explanation is incomplete.
Chaining should be done on KNOWING-THAT15.

Found PLANFOR8 as an explanation for KNOWING-THAT15.
Synopsis:
  Goal: Achieve X;
  Step 1: Know how to X;
  Step 2: Do X
Inferred goal is CHANGE-EVENT3.

The next goal, CHANGE-EVENT3 *(which represents changing the file a.out), is not an instrumental goal, nor is there a strong expectation for a particular action. However, the goal of changing an executable file arouses interest in* PAGAN*; thus, chaining continues:*

Determining whether to chain on CHANGE-EVENT3.
    The explanation is not necessarily complete.
    The explanation is not necessarily incomplete.

```
    Determining the level of curiosity about CHANGE-EVENT3.
        Determining the level of interest in CHANGE-EVENT3.
        Interest level = 4.
        Determining the level to which the goal CHANGE-EVENT3 conflicts
          with system goals.
        Conflict level = 0.
    Curiosity level = 4.

    Determining estimated cost of chaining on CHANGE-EVENT3.
    Estimated cost = 1.
Chaining should be done on CHANGE-EVENT3.

Found PLANFOR9 as an explanation for CHANGE-EVENT3.
Synopsis:
  Goal: change program effect;
  Step 1: Change the program's binary file
Inferred goal is CHANGE-PROGRAM-EFFECT0.
```

The newly-inferred goal is the goal of changing the effects of the program. This goal does not arouse curiosity in PAGAN, *so chaining stops:*

```
Determining whether to chain on CHANGE-PROGRAM-EFFECT0.
    The explanation is not necessarily complete.
    The explanation is not necessarily incomplete.

    Determining the level of curiosity about CHANGE-PROGRAM-EFFECT0.
        Determining the level of interest in CHANGE-PROGRAM-EFFECT0.
        Interest level = 0.
        Determining the level to which the goal CHANGE-PROGRAM-EFFECT0
          conflicts with system goals.
        Conflict level = 0.
    Curiosity level = 0.

    Determining estimated cost of chaining on CHANGE-PROGRAM-EFFECT0.
    Estimated cost = 1.
Chaining should not be done on CHANGE-PROGRAM-EFFECT0.
```

5. External control of chaining

The amount of work that the plan recognizer does on an explanation of an utterance can also be controlled by outside processes. This is achieved by allowing other processes to re-invoke the plan recognizer on an utterance, indicating that more of an explanation of the utterance is needed. Consider the question:

USER: How do I edit one of Barney's files?

The plan recognizer's processing of this question is initially simple. It starts by inferring that the user wants to edit a file owned by Barney. At this point, the plan recognizer determines that the goal it has inferred is neither obviously instrumental, nor intrinsically interesting (the system does not check for potential goal competition between two users). Therefore, the plan recognizer halts its processing of the utterance. UC then calls the KIP planner (Lur88) to plan for the goal of editing Barney's file. KIP tries to do so, but fails. The reason is that, in UNIX, typically only the owner of a file can edit that file. Instead of exerting great energy to find *some* convoluted plan to achieve this goal, KIP simply re-invokes the plan recognizer, indicating that it would prefer to plan for some other goal. It is easy for the plan recognizer to find a parent goal, namely that the user wants a file with contents similar to those of Barney's file.[6] KIP now has a goal with a well-known plan, and so produces the 'copy file, then edit the copy' plan. See Wu (Wu91) for an example of another system that requires this kind of suspension of inference.

This complicated flow of control is advantageous for two reasons. First, when a plan for the initial goal cannot easily be created, the savings in planning time can be considerable. In this example, if KIP worked hard, it might be able to construct a plan for the goal initially suggested by the plan recognizer (for instance, 'boot the machine that is acting as the file server single user, login as root, then edit the file'). Or, KIP might continue planning *ad infinitum* for a goal that it cannot solve. If the plan recognizer can infer a parent goal for which it is easy to construct a plan, as in this example, such wasted planning can be avoided.

The second benefit of this control structure is that in cases where the initial goal can be planned for easily, plan recognition time is reduced. This savings can be significant; continuation of plan recognition, once a reasonable goal has been inferred, is often expensive.

Processing Example

This trace shows how PAGAN can be re-invoked on an analysis that it had initially terminated. The utterance being processed is:

USER: How do I edit Barney's file?

The analysis proceeds until the goal of editing the file is inferred, at which point PAGAN terminates its processing. PAGAN is then re-invoked to continue its analysis. It goes on to infer the goal of obtaining a changed

[6] This is not a parent goal in every context, but making that determination is a separate issue.

version of the contents of the file. The chain of inferred goals in this example is:

1.	TELL-ACTION34:	Direct interpretation of utterance
2.	KNOWING-THAT13:	User know how to change the file
3.	EDITING3:	User change the file
after reinvocation:		
4.	OBTAIN-CHANGED-CONTENTS0:	User have file with modified contents

---

USER: How do I edit Barney's file?
Interpretation produced by the parser: TELL-ACTION34

Chaining is continued on this utterance until the domain goal of changing Barney's file (which is an EDITING by definition) has been inferred:

Determining whether to chain on TELL-ACTION34.
    The explanation is not necessarily complete.
    The explanation is incomplete.
Chaining should be done on TELL-ACTION34.

Found PLANFOR4 as an explanation for TELL-ACTION34.
Synopsis:
  Goal: know something;
  Step 1: ask;
  Step 2: be told.
Inferred goal is KNOWING-THAT13.

Determining whether to chain on KNOWING-THAT13.
    The explanation is not necessarily complete.
    The explanation is incomplete.
Chaining should be done on KNOWING-THAT13.

Found PLANFOR8 as an explanation for KNOWING-THAT13.
Synopsis:
  Goal: Achieve X;
  Step 1: Know how to X;
  Step 2: Do X
Inferred goal is EDITING3.

PAGAN *now determines whether it should try to explain why the user might want to edit Barney's file (as represented by* EDITING3*). It decides against continuing its processing, because the estimated costs of continuing outweigh its interest in continuing. Consequently, processing is halted:*

Determining whether to chain on EDITING3.
    The explanation is not necessarily complete.
    The explanation is not necessarily incomplete.
    Determining the level of curiosity about EDITING3.
    Curiosity level = 0.
    Determining estimated cost of chaining on EDITING3.
    Estimated cost = 1.
Chaining should not be done on EDITING3.

PAGAN *is now re-invoked by an outside process. It continues its processing of the utterance where it left off.* PAGAN *finds the goal of obtaining access to the altered contents of the file as an explanation of* EDITING3. *This goal is easier to plan for than the goal of editing someone else's file; one can simply copy the file, then edit the copy:*

Re-invoking PAGAN to continue its analysis.
Found PLANFOR10 as an explanation for EDITING3.
Synopsis:
  Goal: have file with changed contents;
  Step 1: edit file
Inferred goal is OBTAIN-CHANGED-CONTENTS0.

Determining whether to chain on OBTAIN-CHANGED-CONTENTS0.
    The explanation is not necessarily complete.
    The explanation is not necessarily incomplete.
    Determining the level of curiosity about OBTAIN-CHANGED-CONTENTS0.
    Curiosity level = 0.
    Determining estimated cost of chaining on
      OBTAIN-CHANGED-CONTENTS0.
    Estimated cost = 1.
Chaining should not be done on OBTAIN-CHANGED-CONTENTS0.

---

6. Analysis

In this section, I analyze the need for an inference control mechanism, and assess the performance of the method described here.

6.1. The need for a control mechanism

Inferring the goals that motivate a particular utterance is problematic when the number of goals inferred multiplied by the amount of time required to process each inferred goal is greater than the amount of time the system can afford to spend on plan recognition. This will be the case when one or more of the following conditions hold:

1. The branching factor (that is, the number of plan schemas applicable at each point) is large.
2. The amount of processing required to process each inferred goal is large.
3. Disambiguation cannot be done early, or it cannot be done at all.
4. The depth of the inferred goal chain is large.

Condition 1 is dependent upon the problem domain, and on the degree to which the plan schemas known to the system reflect that domain. Condition 2 is completely system-dependent. Condition 3 represents an open problem toward which a great deal of research is being directed (see for example Appelt and Pollack (AP91), Calistri-Yeh (CY91), London (Lon92), Mayfield (May89), Raskutti and Zukerman (RZ91), and Retz-Schmidt (RS91)).

The aim of the work presented here is to prevent Condition 4 from occurring. The likelihood that this condition will hold in the absence of a method to control inference varies from system to system. For systems that require only simple analyses, such as current natural language front ends to databases, the length of an inference chain is not long enough to warrant the use of this approach. In fact, many systems create a tree of all possible inference chains before processing begins (see for example Sider and Burger (SB92) and Retz-Schmidt (RS91)). As the plan recognition requirements of natural language processing systems increase though, the approach to the control of inference outlined here will become more useful. The UNIX consulting domain seems to have the minimum level of complexity for which this approach is profitable. Inference chains required by the UNIX Consultant average a depth of about 4. With branching factors of about 3, this approach leads to significant savings when it reduces the depth of most chains by even one.

6.2. Performance

An approach to controlling inference in plan recognition is a good one if it meets the following three criteria:

1. It reduces the amount of computation required.
2. It always returns an explanation chain that includes those goals most appropriate to the system's task.
3. It is easy to implement.

The approach to control of inference presented here meets these criteria to varying degrees, as described in the following subsections.

6.2.1. Computational savings

The algorithm presented successfully reduces processing requirements by limiting the length of the inference chains it produces. The algorithm guarantees that no more than one goal outside of the domain of the system will be inferred. In many cases, it limits the amount of processing performed within the domain. Most importantly, it takes into consideration an explicit estimate of the amount of processing that will be required to complete an explanation. While PAGAN's implementation of this estimate is simplistic, the principle of making such an estimate is valuable.

A comparison between system performance with and without the curiosity and difficulty heuristics demonstrates the ability of these heuristics to conserve computational resources. On a test-bed of a dozen questions and short dialogues, the heuristics limited the total number of inferences to 53. Without the heuristics, when chaining was always continued until the inferred goal was outside of the domain, 68 inferences were made. Thus, the heuristics reduced the number of inferences required by 22%.

6.2.2. Explanation quality

A chaining control mechanism must not simply save computational resources, it must do so while still producing explanations that are adequate for the system's purposes. The degree to which a system meets Criterion 2 is difficult to gauge empirically, since it is unclear how one would formalize the notion 'goals appropriate to the system' for unconstrained domains (Mayfield (Mayar) gives a further discussion of the applicability of an explanation to the needs of the system). The algorithm presented here guarantees that at least one domain goal will be inferred. Furthermore, chaining within the domain is done when the user's plans might affect the system's goals in some way. These facts bolster the anecdotal evidence that PAGAN usually produces explanations that subjectively appear to be adequate to UC's needs.

6.2.3. Ease of implementation

The third criterion, ease of implementation, is met with some success by the PAGAN system. PAGAN demonstrates that this approach is viable for a domain of medium complexity such as UNIX consultation. However, PAGAN's implementation details raise the question of scalability. Although this problem does not directly impact the validity of the algorithm (since there are many ways to implement each of the tests required by the algorithm), nonetheless it deserves attention. Of greatest concern is the fact that PAGAN's implementation requires the following three tasks to be done by hand:

1. Categorization of goals as instrumental, domain, or outside-of-domain.
2. Specification of interesting concepts (in cases where goal relationship detection is insufficient or should be compiled).
3. Entry of plan schemas.

Categorization of goals poses no great problem to scalability, primarily because most goals will receive a category automatically via inheritance. Specification of interesting concepts is required in PAGAN primarily because goal relationship detection is implemented only for goal competition with system goals. Detection of the entire range of goal relationships identified by Wilensky (Wil78) will greatly reduce the number of concepts that need to be explicitly marked as interesting. By far, the most problematic of these three tasks is the entry of plan schemas. This is a well-known problem that is not likely to be solved until sophisticated learning algorithms are developed for plan schema acquisition.

7. Conclusion

The availability of a wide variety of heuristics to solve a particular problem often signifies the need for an overall framework in which to organize them. In this paper, I have tried to create just such a framework for chaining heuristics. This framework is centered on an assessment of whether the explanation under consideration is *complete*. If the explanation is complete, chaining is stopped. If the explanation is incomplete, then plan recognition is continued. However, if it cannot be determined whether the explanation is complete, then the system must weigh the strength of its interest in continuing the analysis against the estimated cost of doing so.

This algorithm rests on the principle that the chaining decision should be based primarily on the explanation chain itself, not on external factors. Only when an analysis of the explanation chain proves inconclusive should outside factors weigh heavily in the decision. Furthermore, a decision to discontinue chaining should never be final. Other components of the system should have the opportunity to request that an explanation chain be extended.

Acknowledgements

This work was sponsored in part by the Defense Advanced Research Projects Agency (DoD) monitored by Space and Naval Warfare Systems Command under Contract N00039-88-C-0292; and through the Office of Naval Research, under Contract N00014-80-C-0732. I would like to thank Robert Wilensky for his numerous comments on a previous version of this work.

References

James F. Allen. *A Plan-Based Approach to Speech Act Recognition*. PhD thesis, University of Toronto, Computer Science Department, 1979. Technical Report 131/79.

Douglas E. Appelt and Martha E. Pollack. Weighted abduction for plan ascription. *User Modeling and User-Adapted Interaction*, 1(4), 1991.

Douglas E. Appelt. Planning English referring expressions. *Artificial Intelligence*, 26(1):1–33, 1985.

David Ngi Chin. *Intelligent Agents as a Basis for Natural Language Interfaces.* PhD thesis, University of California at Berkeley, Computer Science Division (EECS), 1988. Technical Report UCB/CSD 88/396.

Randall J. Calistri-Yeh. Utilizing user models to handle ambiguity and misconceptions in robust plan recognition. *User Modeling and User-Adapted Interaction*, 1(4), 1991.

Rhonda Eller and Sandra Carberry. A meta-rule approach to flexible plan recognition in dialogue. *User Modeling and User-Adapted Interaction*, 1(4), 1991.

Diane J. Litman. *Plan Recognition and Discourse Analysis: An Integrated Approach for Understanding Dialogues.* PhD thesis, University of Rochester, Department of Computer Science, 1985. Technical Report TR170.

Robert London. Student modeling to support multiple instructional approaches. *User Modeling and User-Adapted Interaction*, 2(1), 1992.

Marc Luria. *Knowledge Intensive Planning.* PhD thesis, University of California at Berkeley, Computer Science Division (EECS), 1988. Technical Report UCB/CSD 88-433.

James Mayfield. *Goal Analysis: Plan Recognition in Dialogue Systems.* PhD thesis, University of California at Berkeley, Computer Science Division (EECS), 1989. Technical Report UCB 89/521.

James Mayfield. Three properties of a good explanation. In Wolfgang Wahlster, Peter Norvig, and Robert Wilensky, editors, *Intelligent Help Systems for UNIX: Case Studies in Artificial Intelligence.* Springer-Verlag, to appear.

William C. Mann, James A. Moore, and James A. Levin. A comprehension model for human dialogue. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 77–87, 1977.

Peter Norvig. Multiple simultaneous interpretations of ambiguous sentences. In *Program of the Tenth Annual Conference of the Cognitive Science Society*, 1988.

Martha E. Pollack. A model of plan inference that distinguishes between the beliefs of actors and observers. In *Proceedings of the 24rd Annual Conference of the Association for Computational Linguistics*, pages 207–214, 1986.

Alex Quilici, Michael Dyer, and Margot Flowers. Recognizing and responding to plan-oriented misconceptions. *Computational Linguistics*, 14(3):38–51, 1988.

Chuck Rieger. Understanding by conceptual inference. Technical Report TR-353, Computer Science Department, University of Maryland at College Park, 1975.

Gudula Retz-Schmidt. Recognizing intentions, interactions, and causes of plan failures. *User Modeling and User-Adapted Interaction*, 1(2), 1991.

Bhavani Raskutti and Ingrid Zukerman. Generation and selection of likely interpretation during plan recognition in task-oriented consultation systems. *User Modeling and User-Adapted Interaction*, 1(4), 1991.

Roger Schank and Robert Abelson. *Scripts, Plans, Goals and Understanding.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1977.

Judith Schaffer Sider and John D. Burger. Intention structure and extended responses in a portable natural language interface. *User Modeling and User-Adapted Interaction*, 2(1), 1992.

Roger C. Schank. Interestingness: Controlling inferences. *Artificial Intelligence*, 12(3):273–297, 1979.

Candace L. Sidner. Plan parsing for intended response recognition in discourse. *Computational Intelligence*, 1(1):1–10, 1985.

David E. Smith. Controlling backward inference. *Artificial Intelligence*, 39(2):145–208, 1989.

Woo-Young Soh. *Episodic Learning through Conceptual Clustering in a Parallel Activation Model for Diagnostic Reasoning.* PhD thesis, University of Maryland at Baltimore County, Computer Science Department, 1990. Technical Report CS-90-10.

Robert Wilensky, Yigal Arens, and David Chin. Talking to UNIX in English: An overview of UC. *Communications of the ACM*, 27(6):575–593, 1984.

Robert Wilensky, David Chin, Marc Luria, James Martin, James Mayfield, and Dekai Wu. The Berkeley UNIX Consultant project. *Computational Linguistics*, 14(4):35–84, 1988.

Robert Wilensky. *Understanding Goal-Based Stories*. PhD thesis, Yale University, Computer Science Department, 1978. Research Report 140.

Robert Wilensky. *Planning and Understanding: A Computational Approach to Human Reasoning*. Addison-Wesley, Reading, MA, 1983.

Robert Wilensky. KODIAK: A knowledge representation language. In *Program of the Sixth Annual Conference of the Cognitive Science Society*, Boulder, CO, June 1984.

Dekai Wu. Active acquisition of user models: Implications for decision-theoretic dialog planning and plan recognition. *User Modeling and User-Adapted Interaction*, 1(2), 1991.