Analyzing User Plans to Produce Informative Responses by a Programmer's Consultant

CUCS-218-85

Ursula Wolz
Department of Computer Science
Computer Science Building, Columbia University
New York, NY 10027

Abstract

One problem in current help systems for programming utility packages is their inability to provide information within the context of the task at hand. The relationship of a user's goal to the user's plan to accomplish that goal is not taken into account. Furthermore, even a goal/plan based help system should be *informative* by following rules of discourse. This paper puts established rules of discourse into the context of a programming environment, and describes how a report based on an analysis of a user's intended goal and stated plan can be generated. The programming environment under discussion is a programmer's tool kit for graphically exploring complex hierarchical data structures. A program that we have developed called the Plan Analyst is described that finds the relationship between a user's intended goal and stated plan by mapping the goal to the functions of the tool kit. The report generated is informative because discourse rules have been in coded in the knowledge representation that is searched by the Plan Analyst. The intent of this paper is to demonstrate the potential for applying methods of discourse behavior from Natural Language Processing research to an interactive programming environment.

This research was supported in part by the Defense Advanced Research Projects Agency under contract N00039-84-C-0165.

Table of Contents

List of Figures

List of Tables

1 Introduction

Most programming environments include system utilities packages (tool kits) that provide invaluable special features to a programmer. Yet developing the expertise to use them efficiently can often be extremely time consuming. When using a tool kit, it is often the case that a particular task can be accomplished in more than one way, or subtle changes in parameters provide crucial differences in output [Teitelman and Masinter 81]. Documentation cannot possibly cover all of the possibilities, and on-line help often provides insufficient information for the task at hand [Schuster and Finin 83; Finin 83]. One aspect of this problem is the inability of help systems to match the user's plan, that is the functions chosen to do a task, with the user's intended goal. A help system must be able to do more than simply report on how the functions work. It must be able to evaluate the relevance of a user's plan to achieve a goal, and give informative or useful help on why the plan succeeds in meeting or fails to meet the users intended goal.

This paper describes a program called the Plan Analyst that provides relevant information on a utility package in a Lisp programming environment. It uses an and/or graph of plans and goals to develop a mapping from a goal to the functions of the package. The graph also includes information that is required for meaningful explanations about the relationship of the plan and goal. The analysis is developed by building a hypothetical model of the effects of function calls. The program is successful because plans can be instantiated at their lowest level as the functions in the utility package. Furthermore, since the information necessary for proper discourse behavior is encoded in the graph, the report generated by the plan analysis is truly informative.

When using any tool kit, one would like to have access to a human consultant who is well versed in the nuances of the package, and can offer appropriate advice. For example, one might believe that a task can be done using a particular sequence of functions, only to discover upon execution, that something went wrong. A consultant should be expected to respond in a truly helpful way to some form of the following question: "I wanted X to happen, and thought Y would do it, but it didn't -- why not?".

The Plan Analyst can determine the relationship between an intended goal, "X" and a stated plan, "Y" within the domain of a programmer's tool kit for graphically exploring complex hierarchical data structures. The Plan Analyst will serve as a knowledge base search component to a larger automated Consultant System for the tool kit.

The next section describes the tool kit that is the domain on which the Plan Analyst works. It also presents an example of where a user might need help. Section 3 describes the rationale for choosing particular information as relevant to proper discourse based on rules propsed by Proposed, Webber and Weischedel [Joshi, Webber & Weischedel 84]. Section 4 describes and gives examples of how the Plan Analyst determines what that information is.

2 The Domain: A Graphic Display Package for AI Researchers

The Display Tool Kit is being developed to allow researchers in our group to graphically display and manipulate the knowledge structures produced by the generalization programs such as UNIMEM and RESEARCHER [Lebowitz 83]. A clear need was seen for such functions. For example, with a small set of input items, UNIMEM can develop a rather complex structure that occupies pages of line oriented output. UNIMEM generates a tree where the nodes represent generalizations. Links occur when a generalization includes sub-generalizations. Each node is essentially a frame that includes slots that can expand into other complex structures such as discrimination nets [Lebowitz 83]. The tool kit will allow a researcher to interactively get a better feel for the structure of the whole tree, view selected portions of it, display selected portions of a single generalization, and do various kinds of comparisons between generalizations.

One feature of the tool kit is to display the shape of tree structures even when the density of the tree seems to prohibit adequate detail within the bounds of the display. Consequently, there are a number of ways to suppress, highlight and combine portions of the tree. Figures 1 - 3 illustrate the results of using three different methods to view the shape of an entire tree generated by UNIMEM.

In figure 1, a simple method using the default function DISPLAY-AS-MUCH-AS-POSSIBLE is used to produce a tree with all but three very dense sections visible. (These are marked by "#"). A window was created, and its internal size specified. A best fit algorithm was then used to position the nodes of the tree within the dimensions of the window. The algorithm uses a density cutoff parameter to determine when portions of the tree are too dense to fit and marks those portions with a "#". This display might be sufficient for a user new to the system or one who simply wants to confirm that nodes GND2 and GND9 are both two levels below the root.

Figure 2 shows a slightly more sophisticated method where three subwindows with the missing dense portions are positioned in empty parts of the previous display. This display might be sufficient for casually observing the missing portions or perhaps to locate a particular node.

In figure 3, an even more sophisticated method coerces the DISPLAY-AS-MUCH-AS-POSSIBLE function into marking the first level as too dense. The subwindows that were then added give a better visual perspective on the shape of the entire tree. A technique of this kind is required to compare the structural complexity of two subgeneralizations such as GND3 and GND5.

An obvious method that is not shown is to shrink the text size and reduce the density cutoff. In other words, display a tree that extends beyond the bounds of the display, then "shrink" the tree to fit inside the display. This method is possible and is used as an example of a failed plan, since with this particular structure it produced node names that were illegible.

A user who is new to the system might be satisfied with a default display such as 1. A more experienced user might want to try to fiddle with the display to get the whole tree in view. Yet traditional documentation on the functions of the tool kit might not provide information on the *specific technique or plan* needed. Documentation tends to focus on the features of the functions themselves, and not on the goals that can be satisfied by a combination of those functions.

The Consultant System is begin designed to provide the kind of help needed to identify and debug such techniques. For example, a user who is dissatisfied with Figure 2 could ask the Consultant to suggest a better way to display the tree. The Consultant might suggest using the method that produced Figure 3. The Consultant is envisioned as primarily passive, offering advice only when asked to by the user. The Consultant is responsible for maintaining a current model of the interactive environment, including the user's activity and the state of the current display.

The Consultant is also responsible for knowing what can be done within the current environment and how to best answer the user's questions. The Plan Analyst provides the Consultant with the necessary information to tell the user about the feasibility of the plan, why it might have gone wrong, and what other appropriate alternatives

might exist. It accepts an intended goal, a stated plan and the state of the current display from the Consultant. It searches a knowledge base of goals and plans that includes preconditions and effects, along with other information needed by the Consultant for proper discourse behavior. The next section describes what is needed by the Consultant to give an informative response.

3 Requirements for Providing an Informative Response

In order to provide an informative response, the Consultant must know why a particular plan failed. Furthermore, it must be able to make suggestions about how to fix the plan. The Plan Analyst doesn't simply match a plan with an intended goal, it also determines what information is necessary so that the Consultant can be truly helpful in fixing the problem.

Our approach is based on Grice's *Maxims of Quality and Quantity* [Levinson 83] that within the current context one must be truthful and as informative as possible without misleading one's audience. The discourse information produced by the Plan Analyst is based on the formal description presented in Proposed, Webber and Weischedel [Joshi, Webber & Weischedel 84] (JWW) of assertion types needed to respond to users' plan related questions. The intent is to empower the respondent, in this case the Consultant with the ability to provide information, and to avoid producing misleading responses to questions, in this case from the user.

JWW distinguish between the intended goal and the stated goal of the user. The relationship that exists between the two may not be straight forward. For example the intended goal may be more abstract, the stated goal may be an enabling condition of the intended goal, or the intended goal may be a specific case of the stated goal.

We draw a distinction between the intended goal and the plan that is stated to attempt to achieve that goal[1]. The Plan Analyst is responsible for determining whether there is a mapping from the intended goal to the stated plan. In the context of the initial question to the consultant: "I wanted to do X by doing Y", X is the intended goal and Y is the stated plan that achieves it. It should be noted that JWW do not address the problem of how to determine the intended goal from the user, although they point to research that does [Pollack 83]. They assume that the intended goal can be accurately determined. Similarly, the Plan Analyst assumes that the Consultant can

[1] called the stated goal by JWW

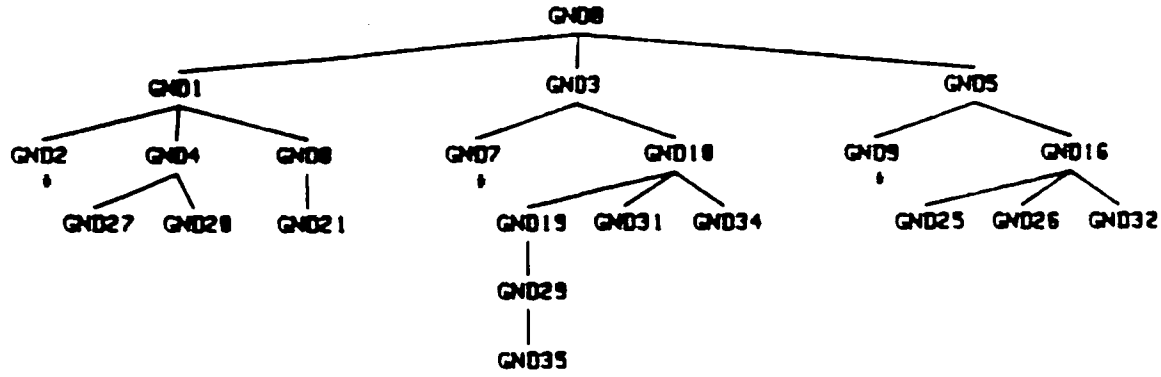

Figure 1: Display As Much As Possible

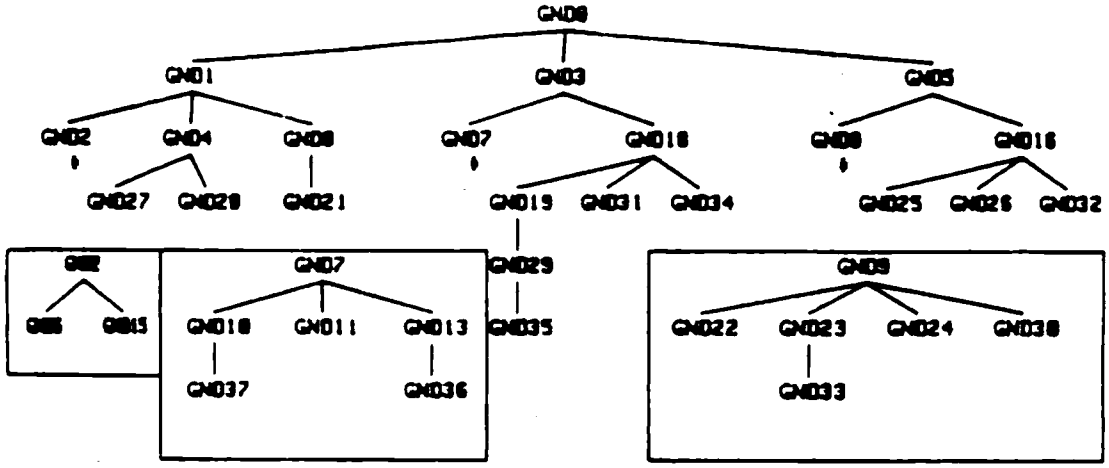

Figure 2: Do Best Fit Then Add Dense Parts

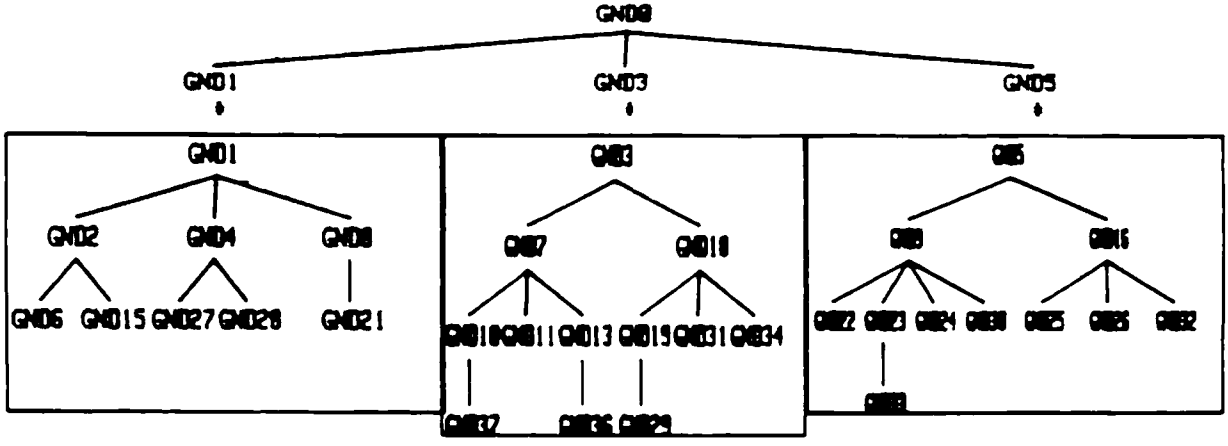

Figure 3: Force Top Level To Punt

accurately determine the intended goal. The stated plan is also received from the consultant. It is simply the list of functions the user indicated were causing problems when requesting help.

JWW also draw a distinction between possible actions and impossible situations[2]. Actions are the execution of plans that can be used to accomplish a goal. At the simplest level they are accomplished by using the functions that are available in the tool package. It is possible at times, that actions cannot be executed because preconditions have not been met. Parts of plans might be inappropriate, for example, plans may contain extra steps that violate preconditions of a later step. Plans may also contain steps in the wrong order where preconditions of early steps are satisfied by steps that appear later. It is also possible that steps are missing from the stated plan, thereby preventing preconditions of later steps from being satisfied.

Impossible situations occur when there is no available action due to system limitations, but in which some action must occur for the goal to be accomplished within the current environment. Impossible situations can be physical system constraints or functions that are not currently available or implemented. For example, there might be a logical sequence of steps that should lead to a particular goal, than cannot be executed because of space and time limitations and visibility constraints of an output device.

Table 1 summarizes what is needed to generate an informative response. The information can fall into one of three classes. If a goal was not achieved due to missing preconditions, then the report should include how the preconditions might be satisfied. The report should include a plan or plans that might satisfy them, or known impossible situations that prohibit them from being satisfied. The report must also be able to indicate that there is no known solution or prohibiting situation, that is it must admit that a solution might be possible, but it has no knowledge of a solution or impossible situation.

If a goal wasn't achieved due to a conflict between the intended goal and stated plan, the report should include the nature of the conflict. The report should distinguish between an inappropriate and incomplete plan. If the plan was inappropriate, that is if a portion of it contained an extra or unnecessary step, then the step and an explanation of why it is inappropriate should be identified. If the plan was incomplete, then the necessary steps to statisfy the goal

[2]Impossible situations are called events by JWW

should be reported.

If the goal was achieved then other relevant information should be reported. Relevant information might include better ways to achieve the goal, or that parts of the plan might have undesirable side effects. Furthermore, it is certainly relevant that a plan is the only known way to achieve a goal, and that information should be reported too.

Even in an environment that contains definitive primitives such as the tool kit functions, the relationship between goals and plans is complex. Any particular goal can be satisfied by a rather vast array of plans, each of which also breaks down into subgoals. Therefore, it is not simply a matter of developing a set of predicates for each goal and plan that are fired when the Plan Anaylyst encounters it. While such a system would work, a more judicious organization and encoding of plans and goals will facilitate the indentification of a conflict, or lack thereof between an intended goal and a stated plan. The next section describes how the Plan Analyst searches a knowledge structure called a Goal Tree to collect the information necessary to make the proper report.

1) The plan does not achieve the goal because preconditions are not met. The report should include:
   (a) A plan that could satisfy them.
   (b) What system limitation might prevent them from being satisfied in the current environment.
   (c) That nothing is known about a possible solution.
2) The plan does not achieve the goal because there is a conflict between the goal and the plan:
   (a) A portion of the plan is inappropriate - an extra or unnecessary step exists.
   (b) The goal is not achieved because a step is missing in the plan.
3) The goal is achieved and:
   (a) There is a better way which should be reported.
   (b) The plan is the only way to achieve the goal.

Table 1: Summary of Rules for Producing an Informative Response

4 The Plan Analyst

The Plan Analyst relies on the fact that a mapping can be made from a high level (intended) goal, to an ordered list of stated plans, which eventually map directly to the functions within the utility package, The functions themselves are the primitives of the plan formalism. The knowledge structure provides the Plan Analyst with the information described in Table 1 to meaningfully report on the feasibility of plans and goals within the interactive environment.

The organization of the representation is a hierarchy of goals, plans and subgoals called a Goal Tree. The

approach used is similar to the programming plan formalism used by Rich for the Programmers Apprentice [Rich 81], and Johnson and Soloway for the programming expert PROUST [Johnson and Soloway 83]. The Goal Tree for each goal contains information on the preconditions and effects of a goal along with any plans and subgoals. Preconditions may include pointers to plans that can satisfy them, or contain information on why, in certain situations they are impossible to satisfy.

The Plan Analyst receives an intended goal, a stated plan and a current state of the display world from the Consultant. As it searches the Goal Tree for the intended goal, it attempts to match the stated plan with plans encoded in the Goal Tree. It checks for precondition satisfaction by building a hypothetical world model from the effects of the plans it explores. The effects are also encoded in the Goal Tree. This section will describe the structure of the Goal Tree in more detail and show how the Plan Analyst can generate reports that satisfy the rules of Table 1.

4.1 A Goal Tree

The Goal Tree consists of a hierarchical organization of goals in the form of an and/or tree. Each goal contains information on preconditions, effects and possible plans to achieve it. Information can also be included on a better way to achieve the goal. Figure 4 illustrates some of the goals that are needed for the goal to display a whole tree.

A goal can have alternative plans, any of which can be a multi-step plan. Furthermore, a goal can be directly achieved by a function of the tool kit. For example, the goal DISPLAY-WHOLE-TREE in Figure 4 can be achieved by any of 4 alternative plans. Note that the goal to display the whole tree can produce a number of different displays such as those in Figures 1 - 3. Therefore alternative plans may produce different results. The nature of those differences is not explicitly encoded, but can be generated by the Plan Analyst by comparing the hypothetical models produced by different plans.

Plans can have multiple steps as illustrated in the goal DO-BEST-FIT-THEN-ADD-DENSE-PARTS. In a set of alternative plans, any one of them can be a multi-step plan, as illustrated in SHRINK-TREE-TO-FIT-FULL-TREE. When a goal appears as an alternative to itself, then there is an explicit function of that name in the tool kit to satisfy that goal. SHRINK-TREE-TO-FIT-FULL-TREE is an example. An alternative multi-step plan can

provide the details of the function. This information is necessary if a user wishes to customize a function. The Consultant can not be of help if there is no plan based knowledge of the definition of the function.

Better ways are explicitly encoded. For example, a better way for DO-BEST-FIT-THEN-ADD-DENSE-PARTS is FORCE-TOP-LEVEL-TO-PUNT. Both of these goals are alternatives to DISPLAY-WHOLE-TREE. For illustration purposes, the criteria for "better" way is that the better way is the least complex. Information could be encoded in the better way that would provide an explanation for *why* it is considered a better way. It is also possible to have better ways for different kinds of users. Rather than simply reporting a single better way, the Plan Analyst would collect any that are found.

Preconditions and effects are associated with goals primarily when the goal can be satisfied directly by a function. They are encoded as predicates that describe possible states of the display environment. Preconditions describe what conditions must be true in order for the function to satisfy the goal. Effects describe the change that occurs to the display environment after the function has been executed.

The model of the display environment is a simple list of such predicates. The Consultant provides the Plan Analyst with a current display model. The Plan Analyst uses that model and adds or deletes the effects of plans it explores to build a hypothetical model. The hypothetical model is used to verify that preconditions of later plans are met.

Some of the preconditions in the Goal Tree include information on plans that may satisfy them. These are encoded as actions. The Plan Analyst reports these to the Consultant as possible ways of meeting failed preconditions. Preconditions can also have impossible situations attached to them. These are currently encoded as text strings that explain why a precondition cannot be met.

The goal SHRINK-TREE-TO-FIT-FULL-TREE is an example of a goal that can be directly satisfied by a function and provides examples of preconditions and effects. The plan is rather straightforward, namely "shrink" the size of the text characters and decrease the density cutoff in order to fit the entire tree within the bounds of the display. In some cases though the tree is so "wide" that the text size must be reduced to a level that is illegible. This was the case in attempting to use this plan on the structure displayed in Figures 1 - 3.

The first precondition to SHRINK-TREE-TO-FIT-FULL-TREE goal states that the density-cutoff must be greater than the length of a node name. Otherwise, the node names cannot be adequately fit into the space that will be allocated for them. The action SHRINK-TEXT-SIZE can be used to reduce the text size so that a node name can fit within the density cutoff. On the other hand, the second precondition states that the text size better not fall below a minimum. There is no action that can fix this problem, since a system limitation has been encountered.

```
GOAL: DISPLAY-WHOLE-TREE
  Preconditions: none
  Effects: none
  Alternative plans: DO-BEST-FIT-THEN-ADD-DENSE-PARTS
                     SHRINK-TREE-TO-FIT-FULL-TREE
                     FORCE-TOP-LEVEL-TO-PUNT
                     DISPLAY-AS-MUCH-AS-POSSIBLE

GOAL: DO-BEST-FIT-THEN-ADD-DENSE-PARTS
  Preconditions: none
  Effects: none
  Multi Step Plan:   DISPLAY-AS-MUCH-AS-POSSIBLE
                     FIT-DENSE-PARTS-IN-EMPTY-SPACES
  Better way: FORCE-TOP-LEVEL-TO-PUNT

GOAL: SHRINK-TREE-TO-FIT-FULL-TREE
  Preconditions:
    (DENSITY-CUTOFF > (LENGTH NODE-NAME))
        Action: SHRINK-TEXT-SIZE
    (TEXT-SIZE > MINIMUM-TEXT-SIZE)
        Impossible Situation:
          "Text size cannot be less than minimum"
  Effects:
     Add:    (TREE-DISPLAYED gnd0)
     Add:    (DENSE-PARTS-EXPANDED yes)
     Delete: (CURRENT-WINDOW-IS-EMPTY)
  Alternative Plans: SHRINK-TO-FIT-FULL-TREE
                     Multi Step Plan: CREATE-A-WINDOW
                                      SHRINK-TEXT-SIZE
                                      DECREASE-DENSITY-CUTOFF
                                      DISPLAY-TREE
```

Figure 4: Portion of the Goal Tree For Displaying the Whole Tree

4.2 The Method Used to Generate an Informative Report

The Consultant provides the Plan Analyst with an intended goal, a stated plan in the form of a list of functions, and a model of the display environment in the form of a list of predicates. The Plan Analyst returns a report that includes at least one of the three cases described in Table 1. The Plan Analyst recursively searches for the functions of the stated plan in the Goal Tree of the intended goal. The Plan Analyst maintains a record of subgoals that were explored, failed preconditions including actions or impossible situations, and steps (functions) that are either extraneous or missing.

Each alternative plan is explored until the functions of the stated plan are exhausted, or some sort of failure is encountered. A plan rarely fails in a unique way. By exploring alternative paths, the Plan Analyst comes up with a number of scenarios relating the plan to the goal. Successful paths take precedence over failed paths, but alternative failed paths have equal weight. It is up to the Consultant to decide how best to communicate this information to the user[3]

As each function is located, preconditions are checked. If they are satisfied, then the effects are added to or deleted from the hypothetical display model. If the function contains a better way, this fact is recorded in the report. If any preconditions fail, they are recorded in the report and the search along the current path is terminated. Similarly, extra or missing steps are recorded, and cause termination of the current search path.

Every goal that is explored is recorded in a list of searched goals. Goals that directly match steps in the stated plan and have preconditions that are satisfied are marked with "[S]". In this way the report contains a history of the goals that were explored. If a plan has no subgoals listed below it, and is not marked by an [S], then it was considered, but not explored in greater detail. This occurs when the next step to be considered in the stated plan fails to match the first plan to the current goal. The Consultant can use this information to reconstruct *where* exactly the plan might have failed.

Figure 5 illustrates how the Plan Analyst can report a failure due to missing preconditions. In order to satisfy the intended goal DISPLAY-WHOLE-TREE, the Plan Analyst explored the subgoal SHRINK-TREE-TO-FIT-FULL-TREE. Since the preconditions for the subgoal were not part of the hypothetical display model, both preconditions failed. The Plan Analyst collected that fact along with a way to satisfy the first one, and an explanation of why the second is an impossible situation. Note that if neither an action or impossible situation is reported, the Consultant can deduce only that the Plan Analyst has no knowledge of either, *not* that neither exists.

Figure 6 shows that an unnecessary step can be identified. REMOVE-WINDOW-FROM-SCREEN is unnecessary to the stated plan. The Plan Analyst explored all of the direct sub goals of DISPLAY-WHOLE-TREE

---

[3]Simple heuristics for plan failure such as number of failures or level at which they occur are less likely to be accurate than failures based on known bug types [Johnson et. al. 83; Brown and Burton 78]. These issues relate to the Consultant not the Plan Analyst, and are beginning to be addressed in our work.

```
Input: Intended Goal = DISPLAY-WHOLE-TREE
       Stated Plan   = SHRINK-TREE-TO-FIT-FULL-TREE
       Actual-World  = (DENSITY-CUTOFF < (LENGTH NODE-NAME))
                       (TEXT-SIZE < MINIMUM-TEXT-SIZE)

Output: Goal Failed = DISPLAY-WHOLE-TREE
        Failed Preconditions:
            (DENSITY-CUTOFF > (LENGTH NODE-NAME)
          A Way: SHRINK TEXT-SIZE
         (TEXT-SIZE > MINIMUM-TEXT-SIZE)
          Impossible Situation: Text size cannot be less than minimum

        Preconditions Failed in: SHRINK-TREE-TO-FIT-FULL-TREE
        Subgoals Searched = DO-BEST-FIT-THEN-ADD-DENSE-PARTS
                            SHRINK-TREE-TO-FIT-FULL-TREE
```

Figure 5: Example of Precondition Failure

looking for INCREASE-DENSITY-CUTOFF as a first plan. It failed to get past the first step in all but FORCE-TOP-LEVEL-TO-PUNT, since it is the only one that contains this step. The path along this plan was successful up to REMOVE-WINDOW-FROM-SCREEN.

The list of subgoals indicates that the search within this alternative goal ceased there. The Plan Analyst returns all of the plans searched, not just the one that succeeded to the lowest level. INCREASE-DENSITY-CUTOFF would have been considered the extra step if it had not been found in FORCE-TOP-LEVEL-TO-PUNT. The Plan Analyst would not have been able to identify which high level plan failed. Consequently, it provides more information by returning the failure points of all paths searched rather than just the most successful.

Figure 7 shows that a missing step can be identified. The goal FORCE-TOP-LEVEL-TO-PUNT requires a step CREATE-WINDOW before DISPLAY-TREE. CREATE-WINDOW is missing from the plan. The Plan Analyst explores all of the high level goals, and is able to get as far as INCREASE-DENSITY-CUTOFF in FORCE-TOP-LEVEL-TO-PUNT As in the previous example, all top level goals are searched.

Figure 8 illustrates that the Plan Analyst can report a "better way" when a plan succeeds. When the goal DO-BEST-FIT-THEN-ADD-DENSE-PARTS is satisfied, the Plan Analyst checked for a better way. It found FORCE-TOP-LEVEL-TO-PUNT. This information was returned along with the state of the hypothetical world model.

To date, we have not encountered a situation were a method for achieving a high level goal is unique.

```
Input: Intended Goal = DISPLAY-WHOLE-TREE
       Stated Plan   = INCREASE-DENSITY-CUTOFF
                       CREATE-A-WINDOW
                       DISPLAY-TREE
                       CREATE-SUB-WINDOW
                       ADD-DENSE-PART-TO-SUB-WINDOW
                       REMOVE-WINDOW-FROM-SCREEN
                       CREATE-SUB-WINDOW
                       .... (more steps) ....
       Actual-World  = .... contains necessary preconditions ....

Output: Goal Failed = DISPLAY-WHOLE-TREE
        Due To: NO GOAL FOR PLAN
        Offending Plan: REMOVE-WINDOW-FROM-SCREEN

        Subgoals Searched = DO-BEST-FIT-THEN-ADD-DENSE-PARTS
                            SHRINK-TREE-TO-FIT-FULL-TREE
                            FORCE-TOP-LEVEL-TO-PUNT
                              INCREASE-DENSITY-CUTOFF [S]
                              CREATE-A-WINDOW. [S]
                              DISPLAY-TREE     [S]
                              FIT-DENSE-PARTS-IN-EMPTY-SPACES
                                CREATE-SUB-WINDOW [S]
                                ADD-DENSE-PART-TO-SUB-WINDOW [S]
                              INCREASE-DENSITY-CUTOFF
                            DISPLAY-AS-MUCH-AS-POSSIBLE
```

Figure 6: Example of An Unnecessary Step in the Plan

```
Input: Intended Goal = DISPLAY-WHOLE-TREE
       Stated Plan   = INCREASE-DENSITY-CUTOFF
                       DISPLAY-TREE
                       CREATE-SUB-WINDOW
                       .... (more steps) ....
       Actual-World  = .... contains necessary preconditions ....

Output: Goal Failed = DISPLAY-WHOLE-TREE
        Due To: NO PLAN FOR GOAL
        Missing Plan: CREATE-A-WINDOW

        Subgoals Searched = DO-BEST-FIT-THEN-ADD-DENSE-PARTS
                            SHRINK-TREE-TO-FIT-FULL-TREE
                            FORCE-TOP-LEVEL-TO-PUNT
                              INCREASE-DENSITY-CUTOFF [S]
                            DISPLAY-AS-MUCH-AS-POSSIBLE
```

Figure 7: Example of an Extra Step in the Plan

However, the Plan Analyst can handle this case by keeping track of when it encounters alternative plans to a goal. If the Plan Analyst never encounters a goal with alternative plans then it only followed a multi step plan and the solution is unique. This information is reported to the Consultant.

```
Input: Intended Goal = DISPLAY-WHOLE-TREE
       Stated Plan   = DO-BEST-FIT-THEN-ADD-DENSE-PARTS
       Actual-World  = .... contains necessary preconditions ....

Output: Goal is Achieved = DISPLAY-WHOLE-TREE

        Subgoals Searched = DO-BEST-FIT-THEN-ADD-DENSE-PARTS
                              Better Way:  FORCE-TOP-LEVEL-TO-PUNT

        Effects on World: (TREE-DISPLAYED gnd0)
                          (BUFFER-EXISTS default-buffer-name)
                          (WINDOW-PARAMETERS deflt-window-params)
                          (DENSITY-CUTOFF default-density-cutoff)
                          (DENSE-PARTS-EXPANDED)
                          (NOT (CURRENT-WINDOW-IS-EMPTY))
```

Figure 8: An Example of a Plan That Has a Better Way

5 Future Work

The Plan Analyst is a procedure for determining how goals and plans relate. It uses an and/or graph of plans and goals that includes plans that can satisfy preconditions, impossible situations that can occur and alternative plans that might be better ways of achieving a goal. The Plan Analyst requires clear identification of both the intended goal and the stated plan. Methods need to be developed for reliably generating these. Furthermore the structure of the display model is currently rather ad hoc. Future work will include creating a more precise representation. In the current implementation, the "better way" for a goal is not clearly defined. We are considering classifying plans in terms of level of expertise, where different alternative plans may be better (or worse) depending on the sophistication of the user. We are also considering ways of encoding an explanation of why a particular plan is considered better in terms of the effects it has.

An interesting side effect of our approach is that the information needed to offer straightforward assistance can be generated by the Plan Analyst. If the user asks: "What does X do", the Consultant simply gives X as both the intended goal and the stated plan to the Plan Analyst. Since the Plan Analyst builds and returns a hypothetical world model, the Consultant can generate an answer based on that model. More importantly the answer would be based on the current actual world in which the user asked the question.

The Plan Analyst has the potential to return a large amount information if the plan is complex. The Consultant can't simply report all of that information to the user. Possibly a second level of analysis needs to be done, comparing the problem state returned by the Plan Analyst with a typical bug corpus [Johnson et. al. 83; Brown and

Burton 78].

At the present time, Goal Trees must be generated by hand. Even in a static environment this is a problem. It seems unlikely that all of the possible goals that can be achieved with a tool kit will be identified in advance. Furthermore the tool kit is intended to be extensible, and information about new user defined functions should be accessible to the Consultant. Consequently there is a need for automatic generation and modification of Goal Trees. We are looking at how generalization methods for hierarchies [Wasserman 85] can be used to assist in this task.

6 Summary

This paper has described how the Plan Analyst locates knowledge about goals and plans in a simple programming environment to assist in debugging. The Goal Tree includes information that can be used to generate responses that maintain principles of proper informing behavior. The representation relies on a hierarchical organization of goals where individual goals include information on their preconditions, effects, plans, subgoals and possible better ways. Furthermore, the preconditions for individual goals include information on what plans will satisfy them, and when they cannot be satisfied. Although the Plan Analyst is only one component of a Consultant system, we believe we have made progress toward developing a comprehensive help system for a programming environment.

Acknowledgments

I would like to thank Prof. Michael Lebowitz for his insightful comments and guidance on the drafts of this paper, and Prof. Kathy McKeown for her valuable help, especially in the early stages of this work, and for suggesting the inclusion of discourse rules in the first place.

References

[Brown and Burton 78]
Brown, J.S. and R.R. Burton.
Diagnostic Models for Procedural Bugs in Mathematics.
Cognitive Science 2:155 - 192, 1978.

[Finin 83] Finin, T.
Providing Help and Advice in Task Oriented System.
In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 176 - 178. International Joint Conference on Artificial Intelligence, Karlsruhe, West Germany, 1983.

[Johnson and Soloway 83]
Johnson W. L. and E. Soloway.
PROUST: Knowledge-Based Program Understanding.
Technical Report YaleU/CSD/RR#285, Yale University, Department of Computer Science, 1983.

[Johnson et. al. 83]
Johnson W. L., E. Soloway, B. Culter and S. Draper.
Bug Catalogue: I.
Technical Report YaleU/CSD/RR#285, Yale University, Department of Computer Science, 1983.

[Joshi, Webber & Weischedel 84]
Joshi A., B. Webber and R. Weischedel.
Living Up to Expectations: Computing Expert Responses.
In *?????*, pages 169 - 175. American Association of Artificial Intelligence, 1984.

[Lebowitz 83] Lebowitz, M.
Concept learning in a rich input domain.
In *Proceedings of the International Machine Learning Workshop*, pages 177 - 182. Champaign-Urbana, Illinois, 1983.

[Levinson 83] Levinson, S.
Pragmatics.
Cambridge University Press, Cambridge, England , 1983.

[Pollack 83] Pollack, M.
Generating Expert Answer Through Goal Inference.
Technical Report Technical Note 316, SRI International, 1983.

[Rich 81] Rich, C.
A Formal Representation For Plans In the Programmer's Apprentice.
In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 1044 -1052. International Joint Conference on Artificial Intelligence, Vancouver, Canada, 1981.

[Schuster and Finin 83]
Schuster. E and T. Finin.
Understanding Misunderstanding - Recognizing and Responding to User Misunderstandings.
Technical Report MS-CIS-83-12, Computer and Info. Science, U. of Penn., 1983.

[Teitelman and Masinter 81]
Teitelman, W and L. Masinter.
The Interlisp Programming Envionment.
In *IEEE Proceedings?????*, pages 25 - 33. IEEE, %%%%%, 1981.

[Wasserman 85] Wasserman, K.
Unifying Representation and Generalization: Understanding Hierarchically Structured Objects.
PhD thesis, Columbia University, Department of Computer Science, 1985.