Expressing Concern

Marc Luria

Computer Science Division
University of California at Berkeley
Berkeley, CA 94720
U.S.A.

Computer Science Department
Technion, Israel Institute of Technology
Haifa
Israel

Abstract

A consultant system's main task is to provided helpful advice to the user. Consultant systems should not only find solutions to user problems, but should also inform the user of potential problems with these solutions. Expressing such potential caveats is a difficult process due to the many potential plan failures for each particular plan in a particular planning situation. A commonsense planner, called KIP, Knowledge Intensive Planner, is described. KIP is the planner for the UNIX Consultant system. KIP detect potential plan failures using a new knowledge structure termed a concern. Concerns allow KIP to detects plan failures due to unsatisfied conditions or goal conflict. KIP's concern algorithm also is able to provide information to the expression mechanism regarding potential plan failures. Concern information is passed to the expression mechanism when KIP's selected plan might not work. In this case, KIP passes information regarding both the suggested plan and the potential caveats in that plan to the expression mechanism. This is an efficient approach since KIP must make such decisions in the context of its planning process. A concern's declarative structure makes it easier to express than procedural descriptions of plan failures used by earlier systems.

1. Introduction

The most important task of a consultant is to provide advice to a user. Human consultants are asked to provide answers to user queries in domains within which they have more expertise than the user. In some cases, the answers provided to the user are basic information about a particular domain. However, in many cases, the task of the consultant is provide answers to user problems. Furthermore, they are not only asked to find solutions, they are also asked to use their expertise to anticipate potential problems with these solutions. Let us consider a very simple example of a consultant relationship. For example, suppose a child asks the following question:

(a) Where is my shoe?

His mother might respond:

(a1) It's in the basement.

However, his mother might also add:

(a2) Let me know if the door is locked.
(a3) Be careful walking down the stairs.
(a4) Make sure to turn off the basement light.

In (a1), the mother has provided the child with information about the location of his shoe. The mother has also implied the use of a plan: Walk down to the basement and get your shoes. However, there are a number of problems inherent in this plan. The mother might also inform her child of these problems. The first problem, (a2), is that one of the conditions necessary to execute the plan might be unsatisfied. The door to the basement might be locked. If it is locked additional steps in the plan will be necessary. The second problem, (a3), is that executing the walk-down-the-stairs plan might result in a fall. The mother knows that this outcome is likely, due to her experience of the child's previous attempts at the walk-down-the-stairs plan. The mother wishes to prevent the child from falling, since this is a potentially dangerous and frightening experience for the child. The third problem, (a4), is that the child might forget to turn off the light in the basement. This would threaten the mothers's goal of preventing the basement light from burning out.

However, the same parent might not add:

(a5) Let me know if the door needs to be oiled
(a6) Be careful walking in the basement
(a7) Make sure to close the basement door

This second set of responses also provide advice that reflects problems due to unsatisfied conditions of the plan or potential goal conflicts. However, the mother might not decide to express these statements to the child since they are either unlikely or unimportant causes of potential plan failure.

Therefore, the mother has made three decisions. First, she has decided which plan to suggest to the child based on his world knowledge. Secondly, she has decided which parts of that plan should be expressed to the child. Thirdly, she has decided which potential caveats in that plan should be expressed to the child based on her experience.

Previous research in intelligent user interfaces (Allen84, Appelt85, McDonald84) has focused on the second decision. Systems attempt not to violate Grice's second Maxim of Quantity: *Make your contribution as informative as is required* (Grice 1975). These systems formulated a response that would provide information or a plan to the user. Allen sought to discover obstacles in the user's plan. He tried to help the user's plan by providing the user with the information he needed to execute that plan. However, he did not provide a mechanism for expressing plan failures. In this paper, we focus on the problem of making decisions regarding those potential problems which should be expressed to the user. However, rather than using a separate mechanism for this purpose, we propose that this decision be made using information provided by the problem solving component of the system, the planner.

We describe a commonsense planner called KIP, Knowledge Intensive Planner. KIP is being developed for UC, the UNIX Consultant system (Luria85, Wilensky 84a, 86). UC provides solutions to user problems in the UNIX operating system domain. KIP provides the information necessary in order to make decisions regarding which potential plan failures should be expressed to the user. KIP must make decisions regarding potential plan failures in order to devise a good plan for the user. Rather than use a separate process to make decisions about those potential plan failures which should be expressed to the user, KIP provides this information to the expression mechanism directly. In the next section, we give some examples of KIP's interaction with expression mechanism. KIP provides information about potential plan failures which will be expressed to the user. We also describe KIP's role in the UNIX Consultant system. In the following section, we describe a declarative representation that allows KIP to detect and provide expression information about potential plan failures.

2. KIP Examples

```
(b) User: How do I print Jim's file on
          the lineprinter?
      UC: Print Jim's file foo by
          typing lpr foo. This plan
          will not work if you don't have
          read permission on Jim's file
          or if the printer is out of
          paper.

(c) User: How do I move a file named
          paul to the file named mary?
      UC: To move the file paul to the

          file named mary, type mv paul
          mary.  However, if the file
          mary exists, it will be
          deleted.

(d) User: How do I change my password?
      UC: To change your passwd use the
          passwd command. However,
          remember that if you change
          your password on one machine,
          it will not be changed on
          other machines.
```

In each of these examples, KIP has selected a known plan for accomplishing the goals of the user. However, in each of these examples, KIP determines that the plan could fail and therefore has decided to express this potential failure to the user.

KIP has a large knowledge-base of information about the UNIX operating system. Decisions regarding UC's own actions are made by UCEgo. The parser and goal analyzer (Mayfield 86) of UC pass KIP a set of goals, and KIP tries to find appropriate plans for those goals. KIP determines a plan for the problem, and notes which potential plan failures should be expressed to the user. KIP passes this decision-making information to the UCExpression mechanism(Chin86, Wilensky86). The expression mechanism decides how to express the plan to the user, given a model of the user's knowledge about UNIX. The plan is then passed to the natural language generator, which generates a natural language response to the user. UC is a conversational system, and if necessary KIP can query the user for more information. Nevertheless, KIP tries to provide the best plan it can with the information provided by user.

3. Concerns

In the previous sections, we have described the importance of informing the user about potential problems with a plan. In this section, we describe a new concept which we have introduced, termed a *concern*. A concern allows KIP to predict potential plan failures and provide knowledge to express potential plan failures to the user.

A concern refers to those aspects of a plan which should be considered because they are possible sources of plan failure. A concern describes which aspects of a plan are likely to cause failure.

There are two major types of concerns, *condition concerns*, and *goal conflict concerns*. These two types reflect the two major types of plan failure. *Condition concerns* refer to those aspects of a plan that are likely to cause plan failure due to a condition of the plan that is needed for successful execution. The conditions about which KIP is concerned are always conditions of a particular plan. (These are fully described in Luria86, 87a).

Goal conflict concerns refer to those aspects of a plan which are likely to cause plan failure due to a potential goal conflict between an effect of a plan and a goal of the user. Goal conflict concerns relate plans to user goals and to other pieces of knowledge that are not part of the plan. Examples of this knowledge include background goals which may be threatened by the plan. Since these background goals are not usually inferred until such a threat is perceived, goal conflict concerns often refer to conflicts between a potential plan and a long-term interest of the user. *Interests* are general states that KIP assumes are important to the user. An interest differs from a goal in that one can have interests about general states of the world, while goals refer to a concrete state of the world. For example, preserving the contents of one's files is an interest, while preserving the contents of the file named file1 is a goal. KIP's knowledge-base includes many interests that KIP assumes on the part of the user. Goals are generated only when expressed by the user, or by KIP itself during the planning process.

Stored goal conflict concerns refer to concerns about conflicts of interest. These are concerns about the selected plan conflicting with an interest of the user. If KIP detects a conflict-of-interest concern, then KIP must determine if it should infer an individual goal on the part of the user that reflects this interest. If KIP decides to infer this individual goal, then a *dynamic concern* between the selected plan and the individual goal is also instantiated. (Goal conflict are described more fully in Luria87b.)

Some plan failures are more likely to occur than others, and some plan failures are more important than others if they do occur. The representation of concerns reflects this difference by assigning a varying degree of concern to the stored concerns in the knowledge base. The degree of a condition concern reflects both the likelihood that the condition will fail, and the importance of satisfying the condition for the successful execution of the plan. There are many factors that determine the degree of concern about a conflict-of-interest. The planning knowledge base designer needs to determine how likely a conflicting effect is to occur, how likely it is that the user holds the threatened goal, and how important this goal is to the user.

In the present implementation of KIP, information regarding concerns of potential plans is supplied by a human expert with a great deal of UNIX experience. Stored concerns are therefore, a way for the planner database designer to express his personal experience regarding those aspects of a stored plan that are most likely to fail. In principle, however, the information might be supplied by an analysis of data of actual UNIX interactions.

4. Concerns and Expression

In this section, we describe the problems that concerns were initially meant to address in plan failure detection. We also describe how this same process has been used to express potential plan failures to the user.

KIP is a a *commonsense planner* (Wilensky83) - a planner which is able to effectively use a large body of knowledge about a knowledge-rich domain. Such knowledge includes a general understanding of planning strategy, detailed descriptions of plans, the conditions necessary for these plans to execute successfully, and descriptions of those potential goal conflicts that the plans might cause. Due to the detailed nature of this knowledge, it is difficult to detect potential plan failures. Condition failures are hard to detect since there are many conditions for any particular plan. Goal conflict failures are difficult to detect since any of the many effects could conflict with any of the many goals of the user. Furthermore, many of the user goals are not inferred until a threat to user interest is perceived. Previous planning programs (Fikes71, Newell72, Sacerdoti74) searched exhaustively among every condition and every potential goal conflict for potential plan failure. This is a very inefficient process. On the other hand, human consultants generally consider only a few potential plan failures while assessing a particular plan.

Additionally, KIP may not be aware of the values of many of the conditions of a particular plan. Most previous planning research assumed that the values for all the conditions is known. However, in UC, when a user describes a planning problem which is then passed to KIP, the values for many conditions are usually left out. All users would believe that normal conditions, like the machine being up, would be assumed by the consultant. A naive user might not be aware of the value of many conditions that require a more sophisticated knowledge of UNIX. An expert user would believe that the consultant would make certain assumptions requiring this more sophisticated knowledge of UNIX. It would be undesirable to prompt the user for this information, particularly for those values which are not important for the specific planning situation.

Therefore, concerns were introduced in order to detect plan failures. Concerns allow KIP to use information about the likelihood and importance of potential plan failures. They allow the planning database designer to store knowledge regarding which conditions are most likely to be unsatisfied, and which goal conflicts are most likely to occur as a result of the execution of a particular plan.

Furthermore, the same concern information can be used in order to determine which potential plan failures should be expressed to the user. When, KIP selects a potential plan, the concerns of that particular plan are evaluated in the particular planning situation. Once the concerns of a plan are evaluated there are three possible scenarios. In the first case, none of the concerns are important in the particular planning situation. The plan is generated to the user without any concern information. In the second case, there is a moderate degree of concern regarding the plan. In this case, the plan is generated along with the concern information. In cases where there is a high degree of concern, the plan is modified or a new plan is selected. These scenarios will be fully explained in the following section. Before describing KIP's algorithm regarding decisions about concerns, we first describe a simple example of the use of concerns. For the purposes of this example, we consider only condition concerns.

5. An Example of the Use of Concerns

The simplest use of concerns addresses the problem of specifying which conditions of a particular plan are important enough invoke the planner's concern. For example, suppose the user asks the following question:

```
(e) How do I print out the file named
    george on the laser printer?
```

KIP is passed the goal of printing the file named george on the laser printer. In this case, KIP's knowledge-base contains a stored plan for the goal of printing a file, namely, the USE-LSPR-COMMAND plan. KIP creates an instance of this plan, which it calls USE-LSPR-COMMAND1. KIP must then evaluate the USE-LSPR-COMMAND1 plan in order to determine if the plan is appropriate for this particular planning situation. This process entails the examination of those conditions likely to cause failure of this plan.

In order to examine these conditions, KIP looks at the stored concerns of the stored plan, USE-LSPR-COMMAND. For each of the stored concerns of the stored plan, it creates a dynamic concern in this individual plan, USE-LSPR-COMMAND1. KIP examines the USE-LSPR-COMMAND plan, and finds that two of its many conditions are cause for concern:

```
(1) the printer has paper
(2) the printer is online
```

The most likely cause of plan failure involves (1), since the paper runs out quite often. Therefore, (1) has a *moderate degree* of concern, and (2) has a *low degree* of concern. KIP considers the most likely concerns first. These concerns are called stored *condition concerns*, because the failure of these conditions often causes the failure of USE-LSPR-COMMAND. KIP therefore creates dynamic concerns regarding the paper in the printer, and the printer being online.

KIP then must evaluate each of these dynamic concerns. In this particular example, there is no explicit information about the paper in the printer or the printer being online. Therefore, KIP uses the default values for the concerns themselves. KIP's concern about paper in the printer is high enough to warrant further consideration. Therefore, this concern is temporarily *overlooked*. However, the concern about the printer being online is *disregarded*. Its degree of concern is low. It is not a very likely source of plan failure. Since there are no other dynamic concerns for this particular plan, KIP looks back at its overlooked concern. Since this is the only concern, and the degree of concern is moderate, KIP decides that this concern should not be *elevated* to a source of plan failure. Rather, KIP decides to express this concern to the user. KIP assumes that, except for this concern, the plan will execute successfully. The plan is then suggested to the user:

```
(E) UC: To print the file george on the
        laser printer, type lpr -Plp
        george. This plan will not work
        if the printer is out of paper.
```

There are many other conditions of the USE-LSPR-COMMAND plan that KIP might have considered. For example, the condition that the file exists is an important condition for the lpr command. However, KIP need not be concerned about this condition in most planning situations, since it is unlikely that this condition will cause plan failure. Hence such conditions are not stored in the long term memory of KIP as stored concerns.

6. KIP's Concern Treatment Algorithm

In the following section, we describe the part of KIP's algorithm that decides what to do with concerns once they have been evaluated. KIP's entire algorithm for determining the concerns of a particular plan is fully described in (Luria86) and (Luria87ab).

Once KIP has evaluated a particular dynamic concern of a particular plan, it can proceed in one of three ways, depending on the degree of that particular concern. If the degree of concern is *low*, KIP can choose to *disregard* the concern. *Disregard* means that the concern is no longer considered at all. KIP can try to modify other parts of the plan, and suggest the plan to the user with no reference to this particular concern.

If the degree of concern is *high*, KIP can choose to *elevate* the concern to a source of plan failure. In this case, KIP determines that it is very likely that the plan will fail. KIP tries to fix this plan in order to change the value of this condition, or tries to find another plan.

The most complex case is when the degree of concern is *moderate*. In this case, KIP can choose to *disregard* the concern, or *elevate* it to a source of plan failure. KIP can also choose to *overlook* the concern.

KIP then evaluates each of the concerns of a particular plan. It addresses all of the concerns which have been elevated to a a source of plan failure. KIP thus develops a complete plan for the problem by satisfying conditions about which it was concerned, and resolving goal conflicts about which it was concerned. Once KIP has developed a complete plan, it is once again faced with the need to deal with the *overlooked* concerns. If the plan will work, except for the overlooked concerns, KIP can again choose to *disregard* the concern. If there are a number of overlooked concerns KIP may choose to elevate one or more of these overlooked concerns to a source of plan failure. The plan is then modified accordingly, or a new plan is selected.

At this point, KIP can also choose to suggest an answer to the user. Any, overlooked concerns are then expressed to the user in the answer.

Furthermore, if the concern has been elevated to a source of plan failure, and no other acceptable plan has been found, KIP can choose to suggest the faulty plan to the user, along with the potential caveats. The concern information is based on default knowledge that assumed by KIP. Therefore, the plans may work if these defaults are not correct even if there are concerns in the particular planning situation. Also, the user may decide that he is not concerned about particular plan failure. For example, KIP may have told the user about a potential deleterious side effect. The user may decide that this side effect is not that important if it occurs. This corresponds to a human consultant, who, when faced with a problem he cannot solve, gives the user a potentially faulty plan with the explanation of the potential caveats. This is more informative for the user than just saying that he doesn't know.

7. Advantages of Concerns

Thus, concerns are used by KIP to decide how the planning process should proceed and how to decide which answer is expressed. In this section, we describe a few more examples of KIP's behavior In these examples, we also refer to a new type of concern called a *violated default concern.* These concerns are accessed by KIP whenever it realizes that a default has been violated. In this way, KIP can use knowledge from *default concerns* when there is no knowledge that defaults have been violated. However, when planning in novel situations, general violated default concerns are accessed. Consider the following examples:

```
(f) How do I edit the file anyfile?
(g) How do I edit Jim's file jimfile?
(h) How do I edit the file groupfile
    which is shared by my group?
```

One of KIP's main concerns in any of the possible editing plans is the write permission of the file. If the user tries to edit a file on which he does not have write permission, the plan will fail. In (f), this concern is inherited from the edit plan with a relatively low degree of concern. According to the default case, the file belongs to the user and he has write permission on the file. Since there is no information about the write permission of the file, the default must be assumed and this concern is *disregarded.* KIP would therefore return a plan of

```
(F) To edit the file named anyfile, use
vi anyfile.
```

In (g), KIP realizes that the default of the file belonging to the user is violated. Due to this default violation, a violated default concern of having write permission on the file is created. This concern of write permission is therefore evaluated by the default mechanism. Since there is a very good chance that the plan will not work, this concern about write permission of the file is *elevated* to a cause of plan failure. Once a condition is a cause of plan failure, KIP must deal with the plan failure. KIP can suggest a plan for changing the condition or try some new plan. In this case, since there is no way to change the write permission of Jim's file, another plan is chosen.

```
(G) In order to edit Jim's file, copy the
file to your directory and then use vi
filename to edit the file.
```

In (h), KIP also realizes that the default of the file belonging to the user has been violated. However, the default value for write permission of this file is different because the file belongs to the user's group. There is a good chance that the user does have write permission on the file. However, since there still is some chance that he does not have group write permission, there is still some concern about the condition. In this case, since the degree of concern is *moderate*, KIP can choose to *overlook* the concern, and suggest the plan to the user. However, the concern is still high enough that the answer expression mechanism (Luria 82ab), might choose to express the concern to the user. The answer to (h) would therefore be:

```
(H) To edit the file groupfile, use vi
groupfile. However, it might not work,
if you don't have write permission on
this particular group file.
```

KIP can therefore use concerns to select a potential plan which has a moderate likelihood of success. KIP can express the plan and its reservations regarding the plan to the user. By temporarily overlooking a concern, KIP may search for other plan failures of a particular plan or other potential plans. KIP can accomplish this without completely disregarding a concern or elevating the concern to a source of certain plan failure.

8. Implementation and Representation

KIP is implemented in Zetalisp on a Symbolics 3670. Concepts are represented in the KODIAK knowledge representation language (Wilensky84b). In particular, knowledge about UNIX commands has been organized in complex hierarchies using multiple inheritance. Therefore, when searching for stored default concerns of a particular plan that uses a particular UNIX command, KIP must search through a hierarchy of these commands. This is also true when looking for default violations. KIP searches up the hierarchy, and retrieves the stored concerns or default violations in this hierarchy.

Stored condition concerns are presently implemented by creating a different CONCERN concept for each concern. Also, a HAS-CONCERN relation is added between each concern and those conditions which are cause for concern. Degrees of concern are implemented by creating a HAS-CONCERN-LEVEL relation between the particular concern and the degree of concern. Degrees of concerns are presently implemented as numbers from one to ten. Dynamic condition concerns are implemented as instances of these stored concerns.

Stored goal conflict concerns are presently implemented by creating a different CONCERN concept for each concern. Also, a 3-way HAS-CONCERN relation is created between each concern, the conflicting effect and the threatened interest or goal which are cause for concern.

Defaults are implemented in the current version of KIP by attaching default values of conditions to the plans themselves. Context-dependent defaults are implemented by exploiting the concretion mechanism of UC, which tries to find the most specific concept in the hierarchy. Therefore, since KIP retrieves the most specific plan in the knowledge-base, it automatically retrieves the most specific defaults.

Violated default concerns are implemented by creating a different VIOLATED-DEFAULT-CONCERN concept for each violated default concern. A HAS-VIOLATED-DEFAULT-

CONCERN relation is added between the concern and the stored default which is violated. Therefore, when KIP has found the default that has been violated, it looks for the violated default concerns that are referenced by this default.

Particular concerns have been entered into the database of UNIX plans through a KODIAK knowledge representation acquisition language called DEFABS. These concerns are all based on my experience using UNIX and on discussions I have had with other UNIX users in our research group. We are currently investigating a way to enter this concern information, using the UCTeacher program (Martin, 1985) a natural language knowledge acquisition system. Eventually, KIP may incorporate a learning component that would allow KIP to detect the frequency of certain plan failures and to store these as concerns.

9. Previous Research

9.1. Planning

Early planners such as STRIPS (Fikes71) did not address Goal Conflict Detection as a separate problem. Conflicts were detected by the resolution theorem prover. The theorem prover compares a small set of add or delete formulas, and a small set of formulas that described the present state and the desired state of the world. If an action deleted the precondition of another action in the plan sequence, backtracking allowed the planner to determine another ordering of the plan steps. ABSTRIPS (Sacerdoti74), modified STRIPS to avoid these interacting subgoal problems by solving goals in a hierarchical fashion. Conflicts in ABSTRIPS were also noticed by the theorem prover. However, since the most important parts of the plan were solved first, they occurred less often and fewer paths were explored. Thus, both these programs identified a plan failure as a failed path in the search tree. Therefore, no information about the nature of a failed path could easily be extracted and expressed to a user of the planning system.

Sacerdoti's NOAH (Sacerdoti77) program separated the detection of conflicts from the rest of the planning process using his *Resolve-Conflicts critic*. This critic detects one particular kind of conflict, in which one action deletes the precondition of another action. We refer to this type of conflict as a deleted precondition plan conflict. The critic resolves the conflict by committing to an ordering of steps in which the action which requires the precondition is executed first. The ordering of steps is usually possible since NOAH uses a *least commitment strategy* for plan step ordering. By separating the detection of goal conflicts from the rest of the planning process, NOAH needs to search fewer plan paths than earlier planners.

In order to detect conflicts NOAH computes a TOME, a table of multiple effects, each time a new action is added to the plan. This table includes all preconditions which are asserted or denied by more than one step in the current plan. Conflicts are recognized when a precondition for one step is denied in another step. In order to construct this table, NOAH must enter all the effects and preconditions for each of the steps in the plan every time a new step is added to the plan.

NOAH'S separation of the Goal Conflict Detection Phase from the rest of the planning process was an important addition to planning research. However, NOAH'S approach is problematic in a number of ways. First, it only detects conflicts that occur as a result of deleted preconditions. Other conflicts, such as conflicts between effects of a plan and other planner goals, cannot be detected using this method. Most of the examples in this paper are part of this category of conflict. If many planner goals were included in a TOME, as would be necessary in real world planning situations, this method would be computationally inefficient. Therefore, the same problems that were discussed earlier in regard to exhaustive search also apply to this method. A TOME is (1) computationally inefficient, (2) not cognitively valid, (3) unable to deal with default knowledge, and (4) assumes that all user goals are known, i.e. would have to evaluate every planner interest in a particular planning situation.

Furthermore, information from a critic which is derived from a TOME is very difficult to express. The only thing that NOAH knows regarding a potential plan failure is that one step in a plan will delete the precondition of another step in the plan. A concern, on the other hand is very easy express to the user. Concerns connect the various objects that are effected by a plan failure. In addition, as in any part of the KODIAK knowledge base, additional expression information can be attached to the concern itself. This difference between a concern and a TOME is another example of the advantage of knowledge-rich declarative representations over procedural representation of knowledge.

9.2. Expression

As discussed earlier, work in intelligent user interfaces(Allen84, Appelt85, McDonald84) has primarily focused on decisions regarding what aspects of a plan should be expressed to the user. Expressing concerns about potential plan failures is a natural extension to these other user interfaces.

The texture of this work is very similar to work done earlier by the author. In earlier work on question answering in a text understanding system (Luria82ab), question-answering was divided into two separate processes. According to earlier work one question-answering process determined what was contained in the answer and how that information was expressed to the user. The first of our two processes determined which part of a causal chain was relevant for a particular answer. The second process determined which part of that causal chain should be generated into a natural language response for the user. This resulted in one relatively simple process that found that correct response, and another more general expression process termed answer expression.

In the present work, the process of expressing potential caveats in a plan was not divided into two new processes. Instead, this process is divided into the preexisting planning component, and a more general expression mechanism. In so doing, we have improved the ability of the planning component to deal with potential plan failures.

10. References

Allen, J. 1984. Recognizing Intentions From Natural Language Utterances. In Michael Brady (ed.) *Computational Models of Discourse* Cambridge, Mass; MIT Press.

Appelt, D. 1982. Planning Natural Utterances to Satisfy Multiple Goals. SRI International AI Center Technical Note 259.

Chin, D. N. 1987. "KNOME: Modeling What the User Knows in UC" to appear in *User Modelling in Dialog Systems*, Springer-Verlag series on Symbolic Computation.

Ernst, G. and Newell, A. 1969. *GPS: A Case Study in Generality and Problem Solving*. New York: Academic Press.

Fikes, R. E., and Nilsson, N. J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, Vol. 2, No. 3-4, pp. 189-208. 1971.

Grice, H. P. Logic and Conversation. In P. Cole (ed.) *Syntax and Semantics, Vol. 3: Speech Acts*, New York: Academic Press, pp. 41-58.

Luria, M. "Question Answering: Two Separate Processes" *Proceedings of the 4th National Conference of the Cognitive Science Society*, Ann Arbor, MI August, 1982.

Luria, M. "Dividing up the Question Answering Process" *Proceedings of the National Conference on Artificial Intelligence*, Pittsburgh, PA. August, 1982.

Luria, M. "Commonsense Planning in a Consultant System" *Proceedings of 9th Conference of the IEEE on Systems, Man, and Cybernetics*, Tuscon, AZ. November, 1985.

Luria, M. "Concerns: How to Detect Plan Failures." *Proceedings of the Third Annual Conference on Theoretical Issues in Conceptual Information Processing*. Philadelphia, PA. August, 1986.

Luria, M. "Concerns: A Means of Identifying Potential Plan Failures." *Proceedings of the Third IEEE Conference on Artificial Intelligence Applications*. Orlando, Florida. February, 1987.

Luria, M. "Goal Conflict Concerns" *Proceedings of the Tenth International Joint Conference on Artificial Inteligence*. Milan, Italy. August, 1987.

McDonald, D. 1984. Natural Language Generation as a computational problem. In Michael Brady (ed.) *Computational Models of Discourse* Cambridge, Mass; MIT Press.

Martin, J., 1985. Knowledge Acquisition Through Natural Language Dialogue, *Proceedings of the 2nd Conference on Artificial Intelligence Applications*, Miami, Florida, 1985.

Mayfield, J., 1986. When to Keep Thinking, *Proceedings of the Third Annual Conference on Theoretical Issues in Conceptual Information Processing*. Philadelphia, PA. 1986.

Newell, A., and Simon, H. A. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, N. J. 1972.

Sacerdoti, E., Planning in a Hierarchy of Abstraction Spaces, *Artificial Intelligence* Vol. 5, pp. 115-135, 1974.

Sacerdoti E. *A Structure for Plans and Behavior* Elsevier North-Holland, New York, N.Y. 1977.

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

Wilensky, R., "KODIAK: A Knowledge Representation Language". *Proceedings of the 6th National Conference of the Cognitive Science Society*, Boulder, CO, June 1984.

Wilensky, R., Arens, Y., and Chin, D. Talking to Unix in English: An Overview of UC. *Communications of the Association for Computing Machinery*, June, 1984.

Wilensky, R., et. al., UC - A Progress Report. University of California, Berkeley, Electronic Research Laboratory Memorandum No. UCB/CSD 87/303. 1986.

Sponsored by the Defense Advanced Research Projects Agency (DoD), Arpa Order No. 4871, monitored by Space and Naval Warfare Systems Command under Contract N00039-84-C-0089.