PROBLEM LOCALIZATION STRATEGIES FOR PRAGMATICS PROCESSING IN NATURAL-LANGUAGE FRONT ENDS*

Lance A. Ramshaw & Ralph M. Weischedel
Department of Computer and Information Sciences
University of Delaware
Newark, Delaware 19716 USA

ABSTRACT

Problem localization is the identification of the most significant failures in the AND-OR tree resulting from an unsuccessful attempt to achieve a goal, for instance, in planning, backward-chaining inference, or top-down parsing. We examine heuristics and strategies for problem localization in the context of using a planner to check for pragmatic failures in natural language input to computer systems, such as a cooperative natural language interface to Unix**. Our heuristics call for selecting the most hopeful branch at ORs, but the most problematic one at ANDs. Surprise scores and special-purpose rules are the main strategies suggested to determine this.

I PRAGMATIC OVERSHOOT AND PROBLEM LOCALIZATION

Even if the syntactic and semantic content of a request is correct, so that a natural language front end can derive a coherent representation of its meaning, its pragmatic content or the structure of the underlying system may make any direct response to the request impossible or misleading. According to Sondheimer and Weischedel (Sondheimer, 1980), an input exhibits pragmatic overshoot if the representation of its meaning is beyond the capabilities of the underlying system. Kaplan (1979), Mays (1980a), and Carberry (1984) have each worked on strategies for dealing with particular classes of such pragmatic failures. This paper addresses the problem of identifying the most significant reason that a plan to achieve a user goal cannot be carried out.

The approach to pragmatic failure taken in this paper is to use a planner to verify the presumptions in a request. The presumptions behind a request become the subgoals of a plan to fulfill the request. Using Mays' (1980a) example, the query "Which faculty members take courses?" is here handled as an instance of an IDENTIFY-SET-MEMBERS goal, and the pragmatics of the query are checked by looking for a plan to achieve that goal. Determining both that faculty members and courses do exist and that faculty members can take courses are subgoals within that plan. A presuppositional failure is noted if the planner is unable to complete a plan for the goal.

Furthermore, information for recovery processing or explanatory responses can be derived directly from the failed plan by identifying whatever blocked goal in the planning tree of subgoals is most significant. Thus, in the example above, if the planner failed because it was unable to show that faculty can take courses, the helpful response would be to explain this presumption failure. We concentrate here on identifying the significant blocks rather than on generating natural language responses.

The examples in this paper will be drawn from a planning system intended to function as the pragmatic overshoot component of a cooperative natural language interface to the Unix operating system. We chose Unix, much as Wilensky (1982) did for his Unix Consultant, as a familiar domain that was still complex enough to require interesting planning. In this system, the pragmatics of a user request are tested by building a tree of plan structures whose leaves are elementary facts available to the operating system. For instance, the following planning tree is built in response to the request to print a file:

```
(PRINT-FILE ?user ?file ?device)
  & (IS-TEXT-FILE ?file)
  & (UP-AND-RUNNING ?device)
  & (READ-PERM ?user ?file)
      | (WORLD-READ-PERM-BIT-SET ?file)
      | (READ-PERM-USER ?user ?file)
          & (IS-OWNER ?user ?file)
          & (USER-READ-PERM-BIT-SET ?file)
      | (READ-PERM-GROUP ?user ?file)
          & (SAME-GROUP ?user ?file)
          & (GROUP-READ-PERM-BIT-SET ?file)
      | (READ-PERM-SUPER-USER ?user)
          & (AUTHORIZED-SUPER-USER ?user)
          & (SUPER-USER-PASSWORD-GIVEN ?user)
```

(The children of AND nodes are preceded by ampersands, and OR children by vertical bars. Initial question marks precede plan variables.) If a single node in this planning tree fails, say (IS-TEXT-FILE ?file), that information can be used in explaining the failure to the user.

* This material is based upon work supported by the National Science Foundation under grants IST-8009673 and IST-8311400.

** Unix is a trademark of Bell Laboratories.

The failure of certain nodes could also trigger recovery processing, as in the following example, where the failure of (UP-AND-RUNNING ?device) triggers the suggestion of an alternative device:

User: Please send the file to the laser printer.
System: The laser printer is down.
Is the line printer satisfactory?

This planning scheme offers a way of recognizing and responding to such temporarily unfulfillable requests as well as to other pragmatic failures from requests unfulfillable in context, which is an important, though largely untouched, problem.

A difficulty arises, however, when more than one of the planning tree precondition nodes fail. Even in a tree that was entirely made up of AND nodes, multiple failures would require either a list of responses, or else some way of choosing which of the failures is most meaningful to report. In a plan tree containing OR nodes, where there are often many alternative ways that have all failed of achieving particular goals, it becomes even more important that the system be able to identify which of the failures is most significant. This process of identifying the significant failures is called "problem localization", and this paper describes heuristics and strategies that can be used for problem localization in failed planning trees.

II HEURISTICS FOR PROBLEM LOCALIZATION

The basic heuristics for problem localization can be derived by considering how a human expert would respond to someone who was pursuing an impossible goal. Not finding any successful plan, the expert tries to explain the block by showing that every plan must fail. Thus, if more than one branch of an AND node in a plan fails, the most significant one to be reported is the one that the user is least likely to be able to change, since it makes the strongest case. (The planner must check all the branches of an AND node, even after one fails, to know which is most significant to report.) For instance, if all three of the children of PRINT-FILE in our example fail, (IS-TEXT-FILE ?file) is the one that should be reported, since it is least likely that the user can affect that node. If the READ-PERM failure were reported first, the user would waste time changing the read permission of a non-text file. Unix's actual behavior, which reports the first problem that it happens to discover in trying to execute the command, is often frustrating for exactly that reason. This heuristic of reporting the most serious failure at an AND node is closely related to ABSTRIP's use of "criticality" numbers to divide a planner into levels of abstraction, so that the most critical features are dealt with first (Sacerdoti, 1974).

The situation is different at OR nodes, where only a single child has to succeed. Here the most serious failure can safely be ignored, as long as some other branch can be repaired. Thus the most significant branch at an OR node should be the one the user is most likely to be able to affect. In our example, READ-PERM-USER should usually be reported rather than READ-PERM-SUPER-USER, if both have failed, since most users have more hope of changing the former than the latter. There is a duality here between the AND and OR node heuristics that is like the duality in the minimax evaluation of a move in a game tree, where one picks the best score at nodes where the choice is one's own, and the worst score at nodes where the opponent gets to choose.

III STRATEGIES FOR PROBLEM LOCALIZATION

Identification of the most significant failure requires the addition to the planner of knowledge about significance to be used in problem localization. Many mechanisms are possible, ranging from fixed, pre-set ordering of the children of nodes up through complex knowledge-based mechanisms that include knowledge about the user's probable goals. In this paper, we suggest a combination of statistical "surprise scores" and special-purpose rules.

A. Statistical Ranking Using Surprise Scores

This strategy relies on statistics that the system keeps dynamically on the number of times that each branch of each plan has succeeded or failed. These are used to define a success ratio for each branch. For example, the PRINT-FILE plan might be annotated as follows:

	SUCCESSES	FAILURES	RATIO
(PRINT-FILE ?user ?file ?device)			
& (IS-TEXT-FILE ?file)	235	3	0.99
& (UP-AND-RUNNING ?device)	185	53	0.78
& (READ-PERM ?user ?file)	228	10	0.96

From these ratios, we derive surprise scores to provide some measure of how usual or unusual it is for a particular node to have succeeded or failed in the context of the goal giving rise to the node. The surprise score of a successful node is defined as 1.0 minus the success ratio, so that the success of a node like IS-TEXT-FILE, that almost always succeeds, is less surprising than the success of UP-AND-RUNNING. Failed nodes get negative surprise scores, with the absolute value of the score again reflecting the amount of surprise. The surprise score of a failed node is set to the negative of the success ratio, so that the failure of IS-TEXT-FILE would be more surprising than that of UP-AND-RUNNING, and that would be reflected by a more strongly negative score.

Here is an example of our PRINT-FILE plan instantiated for an unlucky user who has failed on all but two preconditions, with surprise scores added:

```
                                          SURPRISE
                                SUCCESS/FAILURE SCORE
(PRINT-FILE Ann File1 laser)
  & (IS-TEXT-FILE File1)                  F  -.99
  & (UP-AND-RUNNING laser)                F  -.78
  & (READ-PERM Ann File1)                 F  -.96
      | (WORLD-READ-PERM-BIT-SET File1)   F  -.02
      | (READ-PERM-USER Ann File1)        F  -.87
          & (IS-OWNER Ann File1)          F  -.87
          & (USER-READ-PERM-BIT-SET File1) S +.01
      | (READ-PERM-GROUP Ann File1)       F  -.55
          & (SAME-GROUP Ann File1)        S  +.05
          & (GROUP-READ-PERM-BIT-SET File1) F -.58
      | (READ-PERM-SUPER-USER Ann)        F  -.02
          & (AUTHORIZED-SUPER-USER Ann)   F  -.03
          & (SUPER-USER-PASSWORD-GIVEN Ann) F -.02
```

Note that the success of USER-READ-PERM-BIT-SET is not very surprising, since that node almost always succeeds; the failure of a node like READ-PERM-SUPER-USER, which seldom succeeds, is much less surprising than the failure of UP-AND-RUNNING.

We suggest keeping statistics and deriving surprise scores because we believe that they provide a useful if imperfect handle on judging the significance of failed nodes. Regarding OR nodes, strongly negative surprise scores identify branches that in the past experience of the system have usually succeeded, and these are the best guesses to be likely to succeed again. Thus READ-PERM-USER, the child of READ-PERM with the most strongly negative score, turns out to be the most likely to be tractable. The negative surprise scores at a failed OR node give a profile of the typical success ratios; to select the nodes that are generally most likely to succeed, we pick the most surprising failures, those with the most strongly negative surprise scores.

At AND nodes, on the other hand, the goal is to identify the branch that is most critical, that is, least likely to succeed. Surprisingly, we find that the most critical branch tends in this case also to be the most surprising failure. In our example, IS-TEXT-FILE, which the user can do nothing about, is the most surprising failure under PRINT-FILE, READ-PERM is next most surprising, and UP-AND-RUNNING, for which simply waiting often works, comes last. Therefore at AND nodes, like at OR nodes, we will report the child with the most negative surprise score; at AND nodes, this tends to identify the most critical failures, while at OR nodes, it tends to select the most hopeful. Note that the combined effect of the AND and OR strategies is to choose from among all the failed nodes those that were statistically most likely to succeed.

The main advantage of the statistical surprise score strategy is its low cost, both to design and execute. Another nice feature is the self-adjusting character of the surprise scores, based as they are on success statistics that the system updates on an ongoing basis. For example, the likelihood of GROUP-READ-PERM being reported would depend on how often that feature was used at a particular site. The main difficulty is that surprise scores are only a rough guide to the actual significance of a failed node. The true significance of a failure in the context of a particular command may depend on world knowledge that is beyond the grasp of the planning system (e.g., the laser printer is down for days this time rather than hours), or even on a part of the planning context itself that is not reflected in the statistical averages (e.g., READ-PERM-SUPER-USER is much more likely to succeed when READ-PERM is called as part of a system dump command than when it is called as part of PRINT-FILE). To get a more accurate grasp on the significance of particular failures, more knowledge-intensive strategies must be employed.

B. Special-Purpose Problem Localization Rules

As a mechanism for adding extra knowledge, we propose supplementing the surprise scores with condition-action rules attached to particular nodes in the planning tree. The conditions in these rules can test the success or failure of other nodes in the tree or determine the higher-level planning context, while the actions alter the problem localization result by changing the surprise scores attached to the nodes.

The special-purpose rules which we have found useful so far add information about the criticality of particular nodes. Consider the following planning tree, which is somewhat more successful than the previous one:

```
                                          SURPRISE
                                SUCCESS/FAILURE SCORE
(PRINT-FILE Ann File2 laser)
  & (IS-TEXT-FILE File2)                  S  +.01
  & (UP-AND-RUNNING laser)                S  +.22
  & (READ-PERM Ann File2)                 F  -.96
      | (WORLD-READ-PERM-BIT-SET File2)   F  -.02
      | (READ-PERM-USER Ann File2)        F  -.87
          & (IS-OWNER Ann File2)          F  -.87
          & (USER-READ-PERM-BIT-SET File2) S +.01
      | (READ-PERM-GROUP Ann File2)       F  -.55
          & (SAME-GROUP Ann File2)        S  +.05
          & (GROUP-READ-PERM-BIT-SET File2) F -.58
      | (READ-PERM-SUPER-USER Ann)        F  -.02
          & (AUTHORIZED-SUPER-USER Ann)   S  +.97
          & (SUPER-USER-PASSWORD-GIVEN Ann) F -.02
```

Relying on surprise scores alone, the most significant child of READ-PERM would be READ-PERM-USER, since its score is most strongly negative. However, since IS-OWNER has failed, a node which most users are powerless to change, it is clearly not helpful to choose READ-PERM-USER as the path to report. This is an example of the general rule that if we know that one child of an AND node is critical, we should include a rule to suppress that AND node whenever that child fails. Thus we attach the following rule to READ-PERM-USER:

```
IF (FAILED-CHILD (IS-OWNER ?user ?file))
THEN (SUPPRESS-SCORE 0.8)
```

In our current formulation, the numeric argument to SUPPRESS-SCORE gives the factor (i.e., percentage)

by which the score should be reduced. The rule's affect is to change READ-PERM-USER's score to -.17, which prevents it from being selected.

With READ-PERM-USER suppressed, the surprise scores would then select READ-PERM-GROUP, which is a reasonable choice, but probably not the best one. While the failure of IS-OWNER makes us less interested in READ-PERM-USER, the very surprising success of AUTHORIZED-SUPER-USER should draw the system's attention to the READ-PERM-SUPER-USER branch. We can arrange for this by attaching to READ-PERM-SUPER-USER a rule that states:

```
IF (SUCCESSFUL-CHILD
            (AUTHORIZED-SUPER-USER ?user))
THEN (ENHANCE-SCORE 0.8)
```

This rule would change READ-PERM-SUPER-USER's score from -.02 to -.79, and thus cause it to be the branch of READ-PERM selected for reporting.

While our current rules are all in these two forms, either suppressing or enhancing a parent's score on the basis of a critical child's failure or success, the mechanism of special-purpose rules could be expanded to handle more complex forms of deduction. For example, it might be useful to add rules that calculate a criticality score for each node, working upward from preassigned scores assigned to the leaves. If the rules could access information about the state of the system, they could also use that in judging criticality, so that an UP-AND-RUNNING failure would be more critical if the device was expected to be down for a long time.

C. Other Problem Localization Strategies

While our system depends on surprise scores and rules, an entire range of strategies is possible. The simplest strategy would be to hand-code the problem localization into the plans themselves by the ordering of the branches. At AND nodes, the children that are more critical would be listed first, while at OR nodes, the less critical, more hopeful, children would come first. In such a blocked tree, the first failed child could be selected below each node. A form of this hand-coded strategy is in force in any planner that stops exploring an AND node when a single child blocks; that effectively selects the first child tested as the significant failure in every case, since the others are not even explored. Hand-coding is an alternative to surprise scores for providing an initial comparative ranking of the children at each node, but it also would need supplementing with a strategy that can take account of unusual situations, such as our special-purpose rules.

It might be possible to improve the performance of a surprise score system without adding the complexity of special-purpose rules by using a formula that allows the surprising success or failure of a child to increase or decrease the chances of its parent being reported. While such a formula could perhaps do much of the work now done by special-purpose rules, it seems a harder approach to control, and one more likely to be sensitive to inaccuracies in the surprise scores themselves.

D. Proper Level of Detail

One final question concerns identifying the proper level of detail for helpful responses. The strategies discussed so far have all focused on choosing which of multiple blocked children to report, so that they identify a path from the root to a leaf. Yet the leaves of the planning tree may well be too detailed to represent helpful responses. A selection strategy could report the node containing the appropriate level of detail for a given user. Modeling the expertise of a user and using that to select an appropriate description of the problem are significant problems in natural language generation which we have not addressed.

IV RELATED APPLICATION AREAS

While developed here in the context of a pragmatics planner, strategies for problem localization could have wide applicability. For instance, the MYCIN-like "How?" and "Why?" questions (Shortliffe, 1976) used in the explanation components of many expert systems already use either the already-built successful proof tree or the portion currently being explored as a source of explanations. Swartout (1983) adds extra knowledge that allows the system to justify its answers in the user's terms, but the user must still direct the exploration. An effective problem localization facility would allow the system to answer the question "Why not?"; that is, the user could ask why a certain goal was not substantiated, and the system would reply by identifying the surprising nodes that are likely to be the significant causes of the failure. Such "Why not?" questions could be useful not only in explanation but also in debugging.

In the same way, since the execution of a PROLOG program can be seen as the exploration of and AND-OR tree, effective problem localization techniques could be useful in debugging the failed trees that result from incorrect logic programs.

Another example is recovery processing in top-down parsing, such as using augmented transition networks (Woods, 1970). When an ATN fails to parse a sentence, the blocked parse tree is quite similar to a blocked planning tree. Weischedel (1983) suggests an approach to understanding ill-formed input that makes use of meta-rules to relax some of the constraints on ATN arcs that blocked the original parse. Recovery processing in that model requires searching the blocked parse tree for nodes to which meta-rules can be applied. A problem localization strategy could be used to sort the

list of blocked nodes, so that the most likely candidates would be tested first. The statistics of success ratios here would describe likely paths through the grammar. Nodes that exhibit surprising failure would be prime candidates for meta-rule processing.

Before problem localization can be applied in these related areas, further work needs to be done to see how many of the heuristics and strategies that apply to problem localization in the planning context can be carried over. The larger and more complex trees of an ATN or PROLOG program may well require development of further strategies. However, the nature of the problem is such that even an imperfect result is likely to be useful.

V IMPLEMENTATION DESCRIPTION

The examples in this paper are taken from an Interlisp implementation of a planner which does pragmatics checking for a limited set of Unix-domain requests. The problem localization component uses a combination of surprise scores and special purpose rules, as described. The statistics were derived by running the planner on a test set of commands in a simulated Unix environment.

VI CONCLUSIONS

In planning-based pragmatics processing, problem localization addresses the largely untouched problem of providing helpful responses to requests unfulfillable in context. Problem localization in the planning context requires identifying the most hopeful and tractable choice at OR nodes, but the most critical and problematic one at AND nodes. Statistical surprise scores provide a cheap but effective base strategy for problem localization, and condition-action rules are an appropriate mechanism for adding further sophistication.

Further work should address (1) applying recovery strategies to the localized problem, if any recovery is appropriate; (2) investigating other applications, such as expert systems, backward-chaining inference, and top-down parsing; and (3) exploring natural language generation to report a block at an appropriate level of detail.

VII REFERENCES

Carberry, M. Sandra. "Understanding Pragmatically Ill-Formed Input." Proceedings of the International Conference on Computational Linguistics, 1984.

Kaplan, Samuel J. Cooperative Responses From a Portable Natural Language Data Base Query System. PhD. Dissertation, Computer and Information Science Dept., University of Pennsylvania, 1979.

Mays, Eric. "Correcting Misconceptions About Data Base Structure." Proceedings of the Conference of the Canadian Society for Computational Studies of Intelligence. Victoria, British Columbia, Canada, May 1980, 123-128.

Mays, Eric. "Failures in Natural Language Systems: Applications to Data Base Query Systems." Proceedings of the First Annual National Conference on Artificial Intelligence (AAAI-80). Stanford, California, August 1980, 327-330.

Sacerdoti, E. D. "Planning in a Hierarchy of Abstraction Spaces." Artificial Intelligence 5 (1974), 115-135.

Shortliffe, E. H. Computer Based Medical Consultations: MYCIN. (North-Holland, 1976).

Sondheimer, N. and R. M. Weischedel. "A Rule-Based Approach to Ill-Formed Input." Proceedings of the 8th International Conference on Computational Linguistics, 1980.

Swartout, William R. "XPLAIN: A System for Creating and Explaining Expert Consulting Programs." Artificial Intelligence 21 (1983), 285-325.

Weischedel, Ralph M. and Norman K. Sondheimer. "Meta-Rules as a Basis for Processing Ill-Formed Input." American Journal of Computational Linguistics 9 (1983), to appear.

Wilensky, Robert. "Talking to UNIX in English: An Overview of UC." Proceedings of the 1982 National Conference of Artificial Intelligence (AAAI-82), 103-106.

Woods, William A. "Transition Network Grammars for Natural Language Analysis." Communications of the ACM 13 (Oct. 1970), 591-606.