AN AUTOMATED REASONING TECHNIQUE FOR PROVIDING MOMENT-BY-MOMENT ADVICE CONCERNING THE OPERATION OF A PROCESS

William F. Kaemmerer and James R. Allard

Artificial Intelligence Department
Honeywell Corporate Systems Development Division
1000 Boone Avenue North
Golden Valley, Minnesota 55427

Abstract

A system for continuously providing advice about the operation of some other device or process, rather than just problem diagnoses, must not only function in real time, but also cope with dynamic problem courses. The reasoning technique underlying such a system must not assume that faults have single causes, that queries to the user will be answered and advice to the user will be followed, nor that aspects of a problem, once resolved, will not reoccur. This paper presents a reasoning technique that can be used in conjunction with an inference engine to model the state of a problem situation throughout the entire problem-handling process, from discovery to final resolution. The technique has been implemented and installed on-line in a factory control room, as part of a real time expert system for advising the operators of a manufacturing process.

I. Introduction

There are many potential practical applications for a reasoning technique enabling a knowledge-based system to provide continuous "coaching" to the operators of a complex device or process. For example, manufacturing operations might benefit from advisory systems providing operators with continuous assistance in monitoring and troubleshooting process behavior; similarly, computer installations might provide better service by using expert systems to assist operators in managing the systems' performance. However, the goal of continuously providing a user with operational advice, as well as problem diagnoses, makes unique demands on the reasoning technique to be employed. Such an advisory system must not only function in real time, but also cope with dynamic situations, and unpredictable interactions with the user.

The goal of a real time expert advisory system is not only to monitor the target system to detect, diagnose, and suggest a remedy for problems, but also to *continue* to advise the operator on further actions to take as the problem resolves. Functioning in a dynamic situation requires the ability to revoke or update remedial advice if the corresponding problem resolves of its own accord, or if the remedy is no longer appropriate to the situation. The advisory system also should not rely on assumptions that problems have single causes, or that individual aspects of a problem situation, once resolved, will not reoccur.

The ability for an expert advisory system to function interactively with an operator is required, even if the system matures to the point people are willing to "close the loop" and allow it to exert control over the target system. This is because, in most applications, there will always be some actions that cannot be performed without human intervention (e.g., replacing broken parts, operating manual valves, etc.) Thus, the reasoning technique used by such systems must be able to cope with the unpredictability of operator behavior. The system cannot be based on assumptions that the operator will always approve and comply with recommended actions, respond to queries for information not obtainable through instrumentation, or even be available at the time advice is issued. In many application environments, it is also important that the advisory system not interact with the operator unnecessarily.

This paper presents a reasoning technique we have found suitable for providing the problem-monitoring and the advice-giving functions of a real time, interactive expert advisory system meeting the above requirements.

In related research, Griesmer and others [Griesmer *et al.*, 1984; Kastner *et al.*, 1986] discuss a real time expert system for assisting in the management of a computer installation using the MVS operating system. They describe features added to a forward-chaining inference engine to handle the initiation of actions at the appropriate times, to manage communications among system components, and to exert controls that prevent sequences of remedial actions from being interrupted. However, they do not present methods for interrupting, retracting, or revising advice when it is appropriate to do so, nor for coordinating the treatment of multiple faults arising in the same episode.

A method for reasoning about multiple faults is presented by [deKleer and Williams, 1986]. Their research addresses the problems of coping with very large search spaces comprised of combinations of failed components, and performing diagnostic reasoning from a model of the structure and function of the target system, in a static data environment. Our work focuses on managing diagnostic and remedial efforts over time, in a dynamic environment.

Nelson [Nelson, 1982] has utilized a "response tree" technique as part of an expert system to dynamically select among the possible responses that operators of a nuclear reactor might take in a failure situation. However, the main goal of this approach is to efficiently encode and utilize "precompiled" knowledge about responses that will lead to a safe system shutdown. Our work has been in less critical application domains, and is directed toward methods to help operators keep a system functioning.

The technique we present serves as an adjunct to the inference engine of an expert advisory system, in a similar manner as various Truth Maintenance Systems (TMS, ATMS, etc.) can serve as an adjunct to a deductive reasoner. The latter systems (e.g, [deKleer, 1984]) are used for problems in which the assertions given to the system

are relatively unchanging; the elements of the problem space are inferences based on the givens plus additional assumptions which may later be found incorrect. Thus, dependencies of inferences on assumptions are recorded, so that when a contradiction is discovered, the inferences based on the contradictory set of assumptions are readily identified, and may be explicitly or implicitly "undone." Our technique is appropriate for problems in which the assertions (data) given to the system change frequently; the elements of the problem space are states of affairs that are causally related, but which may or may not hold given the next round of data. Thus, dependencies of the current problem state on the state of antecedent causes are recorded, so that when the status of a cause changes, the effect on the overall course of the problem episode is readily identified.

II. Problem Analysis

Consideration of the type of behavior desired from an expert advisory system leads to several conclusions about the required features of the reasoning technique and knowledge representation to be used. Because the reasoning is to be performed in real time, and is to be about the status of a dynamic target system, the reasoning approach must utilize some form of *multi-valued logic.* At a minimum, logic predicates in the system must be permitted to take on an "unknown" value, as well as true/false, whenever the data involved is too obsolete to be considered a valid descriptor of the corresponding aspect of the target system. Likewise, the advisory system cannot halt and await a response from the user when the value of non-instrumented variable is required; hence the reasoning approach must be able to proceed while some data values are unknown.

The reasoning technique must also be *nonmonotonic*, both in what Ginsberg terms the "truth value" (t-nonmonotonic) and "knowledge" (k-nonmonotonic) sense [Ginsberg, 1986]. The world in which an expert advisory system functions is t-nonmonotonic, in that the truth value of conclusions changes over time. For example, problem situations can spontaneously resolve (eg., if a stuck valve frees itself), default assumptions can prove incorrect (eg., a manual valve normally open may have been closed), or the operator of the system can resolve a problem independently of the advisory system. As a result, the reasoning technique must be able to correctly "back up" in the state of affairs concluded.

The advisory system's world is also k-nonmonotonic, because the amount of information known for certain to the system decays over time, as the data on which it is based ages. As a result, reasoning by an expert advisory system must be *interruptable.* The system cannot afford to suspend data scanning for an indefinite period of time until its inference engine reaches conclusions; data scanning and updates must occur regularly. Although data collection and inferencing can proceed in parallel machine processes, the inference engine must operate on a stable "snapshot" of data, in order to ensure that the data it is using, and hence its conclusions, are internally consistent. Thus, it must be possible to interrupt the reasoning process periodically to allow data updates to occur, then resume. Upon resumption, the reasoning process should not necessarily proceed to follow its prior reasoning paths, which may no longer be productive given the new data, nor can it "start over" each time it receives new data, lest it never reach useful conclusions at all, given the time slice it has available.

These considerations suggest that an effective reasoning approach for an advisory system is one based on a representation of the states a problem can attain during the problem-solving process. Transitions among these states should permit the system to proceed despite incomplete data whenever possible, and enable the system to handle "nonmonotonic progress" through the problem-solving states. (By nonmonotonic progress, we mean transitioning that returns to a previously visited state in the path from the start state to the final state in a problem episode.)

Use of a representation of intermediate states in the problem-solving process makes the inference engine interruptable. The reasoning process can be suspended any time the representational structures are in an internally consistent condition. The problem-solving process will be responsive to data changes that occur during the problem-solving, since upon resumption, the next state transitions will be a function of the newly updated data. In contrast, for example, if a backward-chaining inference engine is interrupted for a data update, and its state (goal stack) saved and restored, the "line of reasoning" that the inferencing will initially pursue is still a function of the goal stack alone.

By defining the state transitions in a way that allows transitioning to occur in some parts of the problem despite unknown data values in other parts, the advisory system can proceed to offer some advice to the operator, even though it must await more data to draw conclusions about other aspects of the problem. As a practical matter, we have found that if the application domain involves problems in which the various potential contributors to a problem are weakly connected, (that is, the cause-effect connections from problems to their potential, underlying causes form more of a tree structure than a lattice), the advisory system can use a strict, three-valued logic, and still generate useful advice while some desired data are unknown. Otherwise, it may be necessary to resort to a more complex logic approach, involving guesses and default values that are subject to later belief revision.

Finally, by defining a state transition network that allows cyclic paths to be followed during a problem episode, the t-nonmonotonic nature of problem-solving in dynamic situations (e.g., the possibility that a subproblem will reoccur within a given overall problem episode) is represented.

III. Technique Used

The "problem status monitoring system" (PSMS) we have developed is for use in conjunction with an inference engine capable of detecting problem conditions and incrementally generating the search space of possible antecedent causes. These antecedent causes are the nodes of the search space; each node has associated with it a single state label from the set defined in PSMS (see below). We assume that the descendants of any given node in the search space, if found to be an actual cause of the current problem, must be remedied or otherwise rendered harmless before their ancestors can be remedied.

The PSMS approach is based on an augmented transition network, consisting of a set of state labels applied to each node of the search space as the problem-solving progresses, and lists attached as properties of each node. The lists are used to record the status of the problem-solving (and remedying) with respect to that node's descendants. Problem nodes transition from state to state depending upon data, the knowledge base of the

advisory system, and the status of these property lists. In turn, state transitions are augmented by actions that update the properties of a node's ancestors in the search space.

A node can be in one and only one state at any given time. The states, and their corresponding labels, are as follows:

nil: No problem-solving from this node has yet begun.

pending: The descendants of this node are under investigation, to be ruled in or out as actual causes of the current problem situation.

diagnosed: At least one of the descendants of this node has been confirmed as a cause of the current problem situation.

ready: All the descendants of this node that were confirmed as causes have been "fixed," hence, the cause represented by this node is ready to be remedied.

no-remedy: One or more descendants of this node has been confirmed as a cause, but no remedy has been effective, and/or no remedy for the cause represented by this node is known.

resolved: The cause represented by this node has been remedied, or ruled out as a contributor to the current problem situation.

uncle: The cause represented by this node has been confirmed as a cause, but no remedy has been found; the advisory system cannot help the user with this aspect of the problem.

Four lists are attached as properties to each node of the problem space. These lists are the list of "confirmed," "rejected," "fixed," and "can't-be-fixed" descendants of the node. If a node is confirmed as a contributing cause of the problem situation, it is entered on its parents' "confirmed" lists. (Note that a node may have more than one immediate parent in the problem space.) Conversely, if the node is rejected as a contributing cause, it is entered on its parents' "rejected" list. Likewise, once a node is confirmed, if the cause it represents in the application domain is remedied, the node is entered on its parents' "fixed" lists. Alternatively, if the advisory system exhausts its supply of recommendations to the user, and the cause remains problematic, the corresponding node is entered on its parents' "can't-be-fixed" lists. The management of these lists obeys the following four constraints:

(1) Set-Union (Confirmed,Rejected) $\subseteq$ {descendants}

(2) Set-Intersection (Confirmed,Rejected) = null

(3) Set-Union(Fixed,Cant-be-fixed) $\subseteq$ {confirmed}

(4) Set-Intersection (Fixed,Cant-be-fixed) = null

The test used to determine the state transition to be undergone by a node in the problem space involves both the advisory system's knowledge base, and the status of these property lists. This transition test consists of a maximum of seven steps, as follows. (The letters in brackets [] correspond to the rows of the state transition table found in Table 1.)

1. The inference engine is called upon to determine whether the problem (cause) represented by node has been remedied; if so [A], the node transitions to Resolved.
2. Otherwise, if new direct descendants of the node can be generated [B], they are added, and the node transitions to Pending.
3. Otherwise, if some of the nodes' descendants are not on either its Confirmed or Rejected lists [C], no transition is made; (the jury is still out on some antecedent causes.)
4. Otherwise, if the nodes' Confirmed list is empty, then if the knowledge base contains some remedial advice associated with this node [D], transition to Ready; else [E] transition to No-Remedy.
5. Otherwise, if not all members of the nodes' Confirmed list are on either its Fixed or Can't-be-fixed lists [F], the node is labeled Diagnosed; (we've confirmed at least one cause, but we're still waiting for some antecedent cause to be remedied).
6. Otherwise, if the nodes' Can't-be-fixed list is not empty [G], and the node is not already labeled No-Remedy, transition to No-Remedy; else, transition to Uncle.
7. Otherwise, if the knowledge base contains some remedial advice associated with this node, transition to Ready [H]; else [I] if the node is not already labeled No-Remedy, transition to No-Remedy, otherwise, transition to Uncle.

By defining the state transition network to include a No-Remedy state as a "way-station" on the way to the Uncle state, a "hook" is provided allowing the advisory system to have a second chance at problem-solving before "giving up." This is useful if an initial attempt at problem solving without involving querying of the user is desirable, to avoid unnecessary interactions with the user. (Specific ways of implementing this approach, and integrating PSMS with the rest of an expert advisory system, are beyond the scope of this paper.)

Table 1 summarizes the PSMS state-transition table. Entries in this table indicate the resulting state that a node assumes, based on its current state (column), and the result of the above test (row). The state transitions are augmented by actions to update the property lists of the nodes' parents. Whenever a node transitions from Pending to Resolved, it is entered on its parents' Rejected lists, as this corresponds to "ruling out" the associated cause as a culprit in the current problem. Whenever a node makes a transition from Pending to any other state except Resolved, it is entered on its parents' Confirmed lists, as it is now known to be a contributor to the problem situation. Similarly, a transition of a node to Resolved from any state (other than Pending) causes it to be entered on its parents' Fixed lists. Any transition to the No-Remedy state causes the node to be entered on its parents' Cant-be-fixed lists. The effect of these actions is to propagate findings about all causes of the problem situation, and readiness for remedial action, from the fringe to the root of the problem search space lattice. To the extent that this lattice is weakly interconnected, progress in problem-solving and advice-giving can proceed along one path from fringe to root, even while other paths are awaiting the results of further data collection and inferencing.

The transition from the Ready state back to itself (row H) is notable. It is here that the advisory system can issue additional advice to the operator regarding how to

remedy the corresponding problem, since presumably any previously issued advice has been ineffective (else the transition in row A, to Resolved, would have occurred).

The ability of PSMS to support nonmonotonic progress in problem-resolution is based on row B of the state transition table. This row indicates that at any point in a problem episode, a node may transition "back" to the pending state. This transition is augmented as follows: When returning to the Pending state, the node is removed from its parents' property lists. If as a result, a parent's Confirmed list becomes empty, that parent transitions to the Pending state, and the updating of property lists proceeds recursively toward the root of the problem lattice. Otherwise, the parent transitions to the Diagnosed state. Unlike the other state transitions in PSMS, this series of propagated transitions must be uninterrupted in order for the representation to be internally consistent. (Otherwise, for example, the parent might remain in a Diagnosed state even though none of its direct descendents are now Confirmed.) However, the propagation may be accomplished in Order(n log n) time, where n is the number of nodes in the problem lattice. Thus, this poses little difficulty for practical real time applications. Of course, if an upper bound for n in the application domain is known, an upper bound for an invocation of PSMS can be determined.

The reasoning technique of PSMS has a type of completeness property that is useful in advisory systems. Assuming that the inference engine it is used with employs a logically complete method for generating the search space and diagnosing individual causes, the PSMS approach assures that if advice to the user is needed and available in the knowledge base, the advice will be issued. Likewise, if no advice for the problem situation exists in

Table 1
PSMS State Transitions from Current State to New State

Results of Transition Test*	Current State: Nil	Pend.	Diag.	Ready	NoRem.	Resol	Uncle
Problem Remedied [A]	Nil	Resol.	Resol.	Resol.	Resol.	Nil	Resol.
New Direct Descendent [B]	Pend.	Pend.	Pend.	Pend.	Pend.	Pend.	Nil
Some Desc. not Conf. or Rej. [C]	**	Pend.	Diag.	**	**	**	**
Confirmed=nil remedy exists [D]	**	Ready	**	Ready	**	**	**
Confirmed=nil no remedy known [E]	**	NoRem.	**	NoRem.	Uncle	**	**
A confirmed desc. not yet fixed [F]	**	Diag.	Diag.	**	**	**	**
Some conf. cause can't be fixed [G]	**	**	NoRem.	**	Uncle	**	**
Conf. desc. fixed remedy exists [H]	**	**	Ready	Ready	**	**	**
Conf. desc. fixed no remedy exists [I]	**	**	NoRem.	NoRem.	Uncle	**	**

* For an interpretation of the row labels, see the text.
** Empty cells are unreachable state/condition combinations.

the knowledge base, the user will be informed of that fact. Justification for these claims follows from inspection of the state-transition network: PSMS will cause the advisory system to generate pertinent advice when it exists, so long as there is no path to the Uncle state for nodes that have advice associated with them except through the Ready state. Table 1 shows there is no path to Uncle except through No-Remedy, and while there are paths into the No-Remedy state from Pending, Diagnosed, and Ready, rows D and H of the table show that there is no path from nodes with advice associated with them to the No-Remedy state except through the Ready state. Similarly, as long as advice to the user is needed (i.e., a problem node hasn't entered the Resolved state), the node will not enter the Uncle state except through the No-Remedy state, at which point the user can be notified that the knowledge base contains no further pertinent advice for the problem.

IV. System Implementation

A PSMS component has been included in a real time expert advisory system we have implemented and installed in the control room of a factory of a major manufacturer of consumer products. The expert system is interfaced to the plant's process control computer, and obtains on-line sensor data from the manufacturing process on a continuous basis. The expert system monitors these data, detects emerging problems with the manufacturing process, and advises the operator regarding actions to take to avoid or recover from them. It then continues to monitor the process, updating and/or retracting advice as the problem situation evolves. The expert system monitors and provides advice on four parallel manufacturing lines, simultaneously.

The system is currently implemented in Zetalisp on a Symbolics computer. The operator interface, data collection component, and inference engine (with embedded PSMS component) run as separate processes, passing messages and data among them. The amount of process data being scanned by the system varies with the state of the manufacturing process; typically, 60-70 data points are being monitored at any given time. Within the inference engine process, the main tasks are emptying the input data buffer from the data collection component, monitoring the manufacturing process for emerging problems, and advancing the problem-solving process (including advancing each problem node through a state transition). On the average, these tasks require 900, 477 and 530 milliseconds, respectively, for a total top-level inference engine cycle of about 2 seconds.

In the manufacturer's application domain, a typical problem search space (lattice) is 2 to 5 plies deep from detected problem to "ultimate" cause. Generating one ply per inference engine cycle, and allowing for the 2 to 3 transitions required for a node to reach the Ready state, the typical amount of processing from problem detection to the first advice to the operator is 4 to 8 inference engine cycles. Thus, if the inference engine had exclusive use of the machine, its "reaction time" to problems would be 8 to 16 seconds. In practice, a multiple second delay was deliberately built into the inference engine cycle to guarantee other processes (operator interface, incremental garbage collection, etc.) ample time to run, yielding a reaction time of about 30 to 60 seconds. This speed is sufficient for the manufacturing application involved.

V. Comments and Conclusion

We have presented a Problem-State Monitoring System, consisting of an augmented transition network of problem states, useful as an adjunct to inference engines for real time expert advisory systems. The defined transitions allow the system to model a real time problem resolution process, even if it follows a nonmonotonic course with subproblems reoccurring in the same episode. Also, the PSMS approach supports the requirement that an advisory system be capable of updating its recommendations in real time, retracting advice that has become unnecessary.

Coupled with the ability interrupt and resume the problem-solving process, the existence of cyclic paths in the transition network allows PSMS to model reoccurring problems. However, this situation also could lead to undesirable cycles in advisory system behavior, with the advisory system repeatedly recommending remedial actions that only temporarily manage a persistent problem. This behavior has not been observed in our application. However, an interesting direction for further research might be to extend the PSMS approach with a meta-level reasoning component to detect cycles and produce advice for resolving the problem on a more permanent basis. Such a system could be one more step toward to goal of a genuinely "expert" assistant to process operators.

References

[deKleer, 1984] J. deKleer. Choices without backtracking. *Proceedings AAAI-84*, pages 79-85, Austin, Texas, American Association for Artificial Intelligence, August, 1984.

[deKleer and Williams, 1986] J. deKleer and B.C. Williams. Reasoning about multiple faults. *Proceedings AAAI-86*, pages 132-139, Philadelphia, Pennsylvania, American Association for Artificial Intelligence, August, 1986.

[Ginsberg, 1986] M.L. Ginsberg. Multi-valued logics. *Proceedings AAAI-86*, pages 243-247, Philadelphia, Pennsylvania, American Association for Artificial Intelligence, August, 1986.

[Griesmer *et al.*, 1984] J.H. Griesmer, S.J. Hong, M. Karnaugh, J.K. Kastner, M.I. Schor, R.L. Ennis, D.A. Klein, K.R. Milliken, and H.M. VanWoerkom. YES/MVS: A continuous real time expert system. *Proceedings AAAI-84*, pages 130-136, Austin, Texas, American Association for Artificial Intelligence, August, 1984.

[Kastner *et al.*, 1986] J.K. Kastner, R.L. Ennis, J.H. Griesmer, S.J. Hong, M. Karnaugh, D.A. Klein, K.R. Milliken, M.I. Schor, and H.M. VanWoerkom. A continuous real-time expert system for computer operations. *Proceedings of the International Conference on Knowledge-Based Systems (KBS-86)*, pages 89-114, London, England, July, 1986.

[Nelson, 1982] W.R. Nelson. Reactor: An expert system for diagnosis and treatment of nuclear reactor accidents. *Proceedings AAAI-82*, pages 296-301, Pittsburgh, Pennsylvania, American Association for Artificial Intelligence, August, 1982.