The Goal/Subgoal Knowledge Representation for Real-time Process Monitoring

James R. Allard[1] **and William F. Kaemmerer**
Artificial Intelligence Department
Honeywell Corporate Systems Development Division
1000 Boone Avenue North
Golden Valley, Minnesota 55427

Abstract

We have developed and implemented a plan representation system which has been used as the knowledge representation for COOKER, a real-time process monitoring and operator advisory system for batch manufacturing processes. This representation (called "Goal/Subgoal" or "GSG") associates two hierarchies of subgoals with each goal: a sequence of subgoals which need to be satisfied to satisfy the superior goal, and a set of requisite subgoals which must remain satisfied throughout the process of satisfying the superior goal. By explicitly representing correct process operating behavior instead of the infinite space of problem behaviors, a broad range of process operation anomalies can be recognized and diagnosed in terms of a single, simple description of the system. In this paper we compare GSG to our first approach at representation, describe the GSG representation, show how goals are used to monitor processes, and describe some results of our installation of COOKER in a manufacturing plant.

I. Introduction

A representation of batch manufacturing processes has been developed, implemented, and installed in a factory as part of a system which uses a goal and subgoal representation to monitor the plant in real-time and provide the plant's operator with advice about its operation. This "Goal/Subgoal", or "GSG" representation uses two hierarchies of subgoals attached to each goal to represent both the sequence of subgoals which need to be satisfied in order to satisfy a superior goal (i.e., the "phases" of a process), and those subgoals which need to remain satisfied throughout the process of satisfying a superior goal (i.e., the "requisites" of a process). The hierarchy of sequenced subgoals is used to represent the "batch" nature of a manufacturing process, and the hierarchy of sets of requisite subgoals is used to represent the "continuous" nature of a process. This representation was adopted from representations in the planning literature.

The GSG approach has several advantages over a knowledge representation scheme we initially used in the project. The initial approach used a set of rules for recognizing phase transitions within the batch manufacturing process, expectations for conditions and events, a set of rules for recognizing problems within phases, a set of diagnostic rules describing problem/cause trees, and another set of rules describing fixes for verified problem/cause tree leaves. In this approach, the primary objects were problems. Conversely, GSG describes the space of behaviors in which the process is working correctly. By focusing explicitly on this space, the system can recognize behaviors falling outside of the process description as problem behaviors. This is an improvement over approaches which try to explicitly describe the infinite space of problem behaviors.

While both approaches are capable of recognizing and diagnosing the same sets of problems, GSG promotes an iterative knowledge engineering approach which first results in a simple knowledge base that recognizes the presence of all problems which affect monitored variables, but diagnoses very few of these problems. Further knowledge base work can then focus on the ability to diagnose a broader range of problem causes. On the other hand, a problem/cause tree approach promotes the generation of a forest of these trees. The resulting knowledge base permits diagnosis of all problems it contains, but it may never reach the point where it covers the full set of problems. The GSG representation also unifies all representation of the manufacturing process into one structure, eliminating the redundancy within the distinct rule sets of the problem/cause tree approach. Another advantage of GSG is that it helps split the representation of process information away from the methodology being used to utilize that information. Splitting these two was especially advantageous for us, since we were concurrently developing the knowledge base and the methodology for applying that knowledge.

The GSG representation is a part of COOKER, an implemented expert system which monitors a batch manufacturing process and provides real-time advice to the operator. COOKER has several functions: It provides process operators with a continuous identification of the current phase of the process. It assists the operator in avoiding and/or recovering from undesirable process conditions by detecting unexpected changes in process variables and informing the operator of them via a textual description. Then, it advises the operator of actions that should be taken to avoid or recover from the problem, indicates the degree of urgency of the advised actions, and provides the operator with a notification when the undesirable process conditions are corrected. On request, the system provides explanations of the rationale behind its suggestions. Finally, if COOKER's resources are insufficient to recommend a safe, appropriate response, it refers the operator to his or her shift coordinator.

Currently, COOKER runs on a Symbolics 3640 Lisp machine connected via an IBM AT microcomputer to a Honeywell TDC 2000 Process Control System and a programmable logic controller. COOKER has four main subsystems: the data frames, data gatherer, operator interface, and inference engine. The latter three subsystems run as concurrent processes. The data gatherer sends data requests to, and buffers replies from the AT. The operator interface manages all windows, displaying advice and questions to the operator,

[1]This report describes work performed at Honeywell. Mr. Allard's current address is Gensym Corporation, 125 Cambridge Park Drive, Cambridge, MA 02140. Please address correspondence to Dr. Kaemmerer.

and receiving replies. The inference engine handles unbuffering data from the data gatherer into data frames, receives replies from the operator, runs the monitoring and problem recognition mechanism on the goals, and runs any problem solving required by the goals.

The balance of this paper provides an overview of COOKER's capabilities, our initial Problem/Cause Tree approach to knowledge representation, the GSG slots and methods for process monitoring and problem recognition, and some conclusions about our system.

II. Initial Approach

A. Domain Features

Upon the initial investigation of the domain of real-time process control advisory systems, several different approaches to knowledge representation seemed attractive to us. Some of our initial explorations are documented in [Kaemmerer and Mawby, 1986]. The representation was built around the concepts of process phases and operator expectations. Since we were dealing with a batch process, there were several different phases of the process, each of which required very different operator actions. Say, for example, we are representing a process for making baked beans. In one phase the operator would fill a pressure cooker with beans, and in the next phase, he or she would heat and pressurize the vessel to cooking levels. During each of those phases, operators have expectations about *conditions* which should hold over the process variables, and expectations about *events* which should occur within some time frame. If an operator's expectation about some condition or event was not met, he or she would recognize that as a problem, or a precursor to a problem, and take action. An example of a *condition expectation* being violated could be the pressure in a pressure cooker rising to a level which could pop open one of its safety valves. An example of an *event expectation* being violated is the temperature of a cooker not rising above some threshold by a certain time during product heatup, showing that the process was behind schedule. We surmised that better and more experienced operators would have more expectations and better recognition of the status of those expectations than novice operators. We held that the following were necessary components for a real-time process monitoring system: phase tracking, condition expectation monitoring, and event expectation monitoring.

B. Problem/Cause Trees

Based on this analysis, we developed a knowledge representation which we called the Problem/Cause Tree approach. It included a set of phase transition rules and expectations as data objects within the system. In support of those mechanisms we had rules which would recognize when an expectation had been violated and would start a problem solving session. Other rules generated and confirmed or rejected possible causes for the problem, and rules associated with each problem/cause tree leaf generated operator advice. Using KEE, a commercial expert system development environment, as a rapid prototyping tool, we made an initial implementation of this system.

As knowledge acquisition and encoding of the received information continued, several problems became apparent. The first was that much of the information we received had to be represented more than once in the knowledge base. This presented itself most notably in the cases of phase transition rules and problem/cause tree rules. Each phase was supposed to take a certain amount of time, and if that time limit was exceeded there was a problem. To represent and identify these problems we wrote event expectations which mirrored most of our phase transition rules, resulting in double representation of a large body of information. Also, since our problem solving method required explicit rejection of causes as possible problem culprits, we needed a positive and negative statement of each problem/cause rule, again resulting in a double representation. More redundancies occurred in the problem/cause trees, since it was difficult to use the information within one tree as branches in different problem trees which shared similar causes. A second problem was the extent to which our methodologies for handling information and doing problem solving were influencing the way our rules were written. We recognized that a change in our methodology would force us to rewrite most of our rules. A third problem surfaced as we tried to extend our coverage of the possible problems in the plant. Using the Problem/Cause Tree approach, it was not easy to see, by inspection of the knowledge base, whether or not a given problem type was completely handled, nor how thorough the coverage was across the range of possible problems.

III. The Goal/Subgoal Approach

These problems with the Problem/Cause Tree approach led us to develop GSG as a method for representing process information. The GSG representation is implemented as a defined flavor in Symbolics Zetalisp. In this object oriented programming scheme, state is retained in slots on each instance of a flavor and operations on instances are provided by methods. Various goal slots contain pointers to other goals, compiled functional objects which implement conditions associated with the goal, or strings describing the goal. Several methods have been defined on goals which implement condition checking, phase transitions, and problem solving. (See [Kaemmerer and Allard, 1987] for a description of the method for *monitoring* progress in problem solving.) The central mechanism for process monitoring is implemented in the method SATISFIEDP. A plant process is represented by a lattice of goals and subgoals. Each goal represents a plan to be carried out and its subgoals are a decomposition of it into subplans. Goals may also have a set of subgoals which represent conditions which must remain satisfied during the attempt to accomplish the superior goal. The current phase of a process is represented via a goal's progress through its sequence subgoals. Each subgoal can have its own subgoals, and record its own progress through them.

A. Goal/Subgoal Slots

The following slots are used to build GSG objects.

Sequence: An ordered list of goals which represents the substeps involved in satisfying this goal. Before a goal can test its success-criterion or preventers to declare itself as satisfied, the goal must determine that each of the subgoals in its sequence list, in turn, have been satisfied. This slot fills the requirement for phase tracking.

Preventers: A set of goals which must be satisfied simultaneously to satisfy a parent goal, after the sequence goals are satisfied. If there is a success-criterion it will be tested instead the preventer goals, but the preventers may still be present and can be used in problem solving, as they represent potential causes of failing to satisfy the goal, if they themselves aren't satisfied.

Success-criterion: A compiled condition which is tested after all sequence goals have been satisfied to see if this goal is now satisfied.

Requisites: A set of goals which represents conditions which are expected to hold throughout the attempt to satisfy this goal. These goals fill the requirement for condition expectations. If a requisite is not satisfied, then the parent goal has a problem. Requisites are checked only if the parent goal itself is not satisfied.

Problem-yet: A compiled condition which is tested if a goal is sent a SATISFIEDP message, and was found to be not satisfied. If this condition returns TRUE, then this goal has a problem. If it returns FALSE, then the lack of satisfaction of this goal is a normal event as we wait for some process to complete. This condition fills the requirement for event expectations.

Text: An English description of the problem that exists if this goal is not satisfied. It is used in status messages to the plant operator.

The subgoals which need to be satisfied to satisfy a superior goal are the sequence subgoals and the set of preventers. These represent the batch nature of a process. The relationship between preventers and the success-criterion is as follows: When both are present, it is intended that the success-criterion should follow from a conjunction of the conditions represented by the preventer goals. Problem solving works by inspecting the set of subgoals of a problem goal which are blocking satisfaction of the superior goal. Thus, the presence of a success-criterion and a set of preventers provides the ability to encode both an absolute test for satisfaction of the superior goal and a set of diagnostic avenues to follow if there is a problem.

An example of a situation where this is useful is a goal for opening some valve A, which has interlocks on its controller requiring valves B and C to be closed, and D to be open. In a goal such as this, the success-criterion would check the limit switch which indicates if valve A is truly open, and preventers of this goal would be made with success criteria that check that valves B and C are closed, and D is open. With this representation, the goal for A will only satisfy if A actually is opened. If any of the valves B, C, or D are in the wrong position, and are preventing A from opening through interlocks, it can be found in the problem solving process by isolating any preventer goals which are not satisfied. Also, if there is a case where B, C, and D are all in their correct positions, yet valve A still does not open, GSG operates correctly by not allowing the goal for A to satisfy, as well as rejecting valves B, C, and D as possible causes of the problem.

B. Goal/Subgoal Methods

There are three methods associated with the goal flavor which perform the operations required to monitor processes representing by goal trees. These methods are ACTIVATE, SATISFIEDP, and DEACTIVATE. ACTIVATE and DEACTIVATE perform initialization and other bookkeeping functions for goals and their subgoals, and SATISFIEDP is used to check if a goal has become satisfied. When a goal receives the ACTIVATE message, it stores the time at which it is being activated, sends the ACTIVATE message to all of its requisite goals, and sets its current position in the list of sequence subgoals to be the head of the list. If the sequence list is not empty, it also sends the ACTIVATE message to the goal at the head of that list. If there are no sequence goals, it sends the ACTIVATE message to all its preventer goals. When a goal receives the DEACTIVATE message, it sends DEACTIVATE to its requisite goals and to a current sequence goal, if any, which is trying to be satisfied. If there is no sequence left, DEACTIVATE is sent to any and all preventer goals. The SATISFIEDP method is described in detail below.

COOKER's inference engine co-process handles GSG objects in the following way. For every manufacturing line to be monitored there is a top level goal. The ACTIVATE message is sent to the top level goal when starting a batch. After the goal representing the process is activated, the inference engine enters a loop in which it unbuffers any data received from the AT into the data frames subsystem, sends each top level goal the SATISFIEDP message, spends time doing any problem solving required, and then waits if it has arrived at the end of the loop before the minimum top level loop time has elapsed. The wait state is entered so that the other processes running on the machine, such as the user interface, the data I/O process, and the garbage collector, will be able to get enough processing time. If the call to SATISFIEDP on the top level goal returns TRUE, then the goal is sent DEACTIVATE and ACTIVATE again to start a new batch.

The SATISFIEDP message is used to ask a goal if its conditions for success have been met, to allow that goal to advance itself through its phases, and to allow it to check any conditions it monitors, possibly declaring that it has a problem. There are three phases to the SATISFIEDP method: advancing, success checking, and condition checking.

1. Advancing

Upon receiving a SATISFIEDP message, a goal advances itself through any remaining sequence subgoals which have not yet been satisfied. If there are none left it goes directly to the success checking phase. If there are some left it advances by sending its next sequence goal a SATISFIEDP message. If the subgoal returns TRUE, the subgoal is sent a DEACTIVATE message, and the current sequence position is set to the next goal down the sequence list, or to NIL if there are no goals left. When there are no subgoals left on the sequence list, this goal proceeds to the success checking phase. If there is another goal on the list, it is sent an ACTIVATE message, and then the superior goal loops back to the top of the advance procedure again and sends the newly activated subgoal a SATISFIEDP message. If any sequence subgoal replies FALSE to a SATISFIEDP message, then the goal will not satisfy, and it enters the condition checking phase.

2. Success Checking

If a goal has succeeded in sequentially satisfying its sequence subgoals, or if there were none to start with, the goal enters the success checking phase in which it checks its success criterion or preventers to see if it can satisfy. In this phase, if a goal has a success-criterion condition, that condition is run and if it returns TRUE then (Hurrah!) the goal will immediately return TRUE in response to its SATISFIEDP message. If the condition returns FALSE, then the goal will not satisfy and it goes to the condition checking phase. If there is no success-criterion, then the goal will check its preventers. If there are some preventer subgoals, each is sent a :SATISFIEDP message. If all return TRUE then this goal is satisfied and returns TRUE. However, if there are no preventers or if one of them returns FALSE, then this goal will not satisfy and will enter the condition checking phase.

3. Condition Checking

If a goal enters the condition checking phase it has already been determined that it will be returning FALSE to the SATISFIEDP message. In this phase it is checking that its expectations, in the form of a requisites list and a problem-yet criterion, are still being met. It begins by sending any and all of its requisite goals a SATISFIEDP message. If any of them return FALSE to the message, then this goal declares that it has a problem, since requisites represent condition expectations which should hold true throughout an attempt to satisfy this goal. Next, if the goal has a problem-yet condition, it tests that condition and if TRUE is returned, then this goal is declared to have a problem. If the problem-yet condition returns FALSE, then it is a normal, acceptable situation that this goal has not yet satisfied, and no problem will be declared. After the goal has or has not been declared a problem, the goal returns FALSE as a response to its SATISFIEDP message.

Summing up, to become satisfied a goal must first sequentially satisfy each of its sequence subgoals. Note that progress made in one response to SATISFIEDP persists to the next response. Next the goal must receive a TRUE response from a success-criterion condition, or if the goal has no success-criterion, it must have at least one preventer, and all its preventers must all be simultaneously satisfied. If a goal is not satisfied, then it will be declared to have a problem if any of its requisite subgoals is not satisfied, or if it has a problem-yet condition and that condition returns TRUE.

Also note that a goal may return TRUE to one call to SATISFIEDP, and FALSE to the next without any intervening calls to DEACTIVATE and ACTIVATE. This feature is needed for goals which are used as requisites. A requisite may be satisfied, not satisfied, and then satisfied again during the course of satisfying its superior goal.

C. An Example

Figure 1 shows a Goal/Subgoal hierarchy which could be used to represent a baked bean cooking process. In the diagram, the very thick arrows represent sequence subgoal links, such as that between Cook A Batch and Load Beans; the thick arrows represent preventer subgoal links, such as that between Loader Locked and Relief Locked; and the thin arrows off the side of a goal box represent requisite subgoal links.

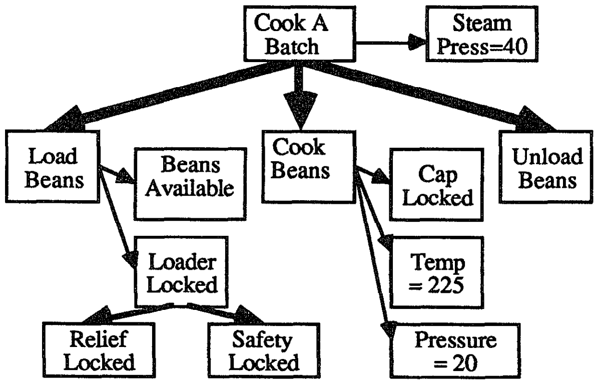

Figure 1: Goal Hierarchy for cooking beans

The following example illustrates the operation of the SATISFIEDP method. Suppose that the cooker has just finished being loaded with beans, its cap has been shut and pressurized, but it has not yet been heated up. The top level loop of COOKER's inference engine has just finished unbuffering data from the data gatherer, and sends the SATISFIEDP message to Cook A Batch. Cook A Batch enters its advancing phase and sends SATISFIEDP to Load Beans. Load Beans goes into its advancing phase, finds that there are no sequence goals, and enters its success checking phase. This goal has a success-criteria which checks a level sensor switch in the cooker which turns off when the beans reach the right level. Load Beans checks its success criterion, the switch is off, the success criterion returns TRUE, and Load Beans immediately returns TRUE. Note that it does not go into its condition checking phase. Its Loader Locked requisite is probably already violated by now, but even if it is violated, it doesn't matter since Load Beans is satisfied. Cook A Batch receives TRUE from Load Beans, sends DEACTIVATE to Load Beans, advances its sequence list, and sends ACTIVATE and SATISFIEDP to Cook Beans. Cook Beans finds it has no sequence list, goes to success checking, finds a success-criterion and calls it. The criterion finds that the beans have not yet been cooking for 2 hours, and returns FALSE. Cook Beans enters its condition checking phase and sends SATISFIEDP to Cap Locked, Pressure=225, and Temp=220, and all but Temp=220 return TRUE. Cook Beans responds by declaring itself a problem, and then returns FALSE to Cook A Batch. Cook A Batch leaves its advancing phase and enters its condition checking phase It sends SATISFIEDP to Steam Press=40, which returns TRUE. It checks its problem-yet slot which checks if more than 4 hours have passed since this cook was started, and it returns FALSE. So, Cook A Batch is not satisfied, but it is not a problem, and it returns FALSE to the top level loop. When problem solving, the system notifies the operator that there is a problem in Cook Beans, and that the cooker is not yet up to temperature.

D. Discussion

Using the information in goal slots and the operations provided by flavor methods, GSG provides the necessary abilities we identified for real-time process monitoring: phase tracking, condition expectation monitoring, and event expectation monitoring. The information needed for phase tracking is stored as a pointer to the current position in a goal's sequence subgoals slot. This information is needed to represent progress through a batch process. The information needed to monitor condition expectations is stored as a set of subgoals in the requisites slot. We think of each phase of a batch process as a continuous process, and this information is needed to recognize problems in continuous processes. The information needed to monitor event expectations is stored as a pointer to a condition in a goal's problem yet slot. This information is used to recognize problems when progressing through a batch process.

GSG's positive statement of the desired behavior of a process makes it easy to recognize a broad range of problem situations. Furthermore, even if a GSG knowledge base is not complete enough to provide a diagnosis of the cause of a problem, it will nevertheless enable the expert system to recognize when a problem exists and alert the operator. This feature aids the quick initial representation of new manufacturing plants and processes for problem recognition, with the ability to incrementally add further diagnostic information at a later date.

GSG's hierarchy of goals stems from plan representations in the literature, such as the one in Chapman's TWEAK [Chapman, 1985]. In TWEAK, steps represent actions, and each step has associated with it a set of preconditions and postconditions. Plans are generated by starting with a goal, which is a desired condition. A temporal ordering on steps is then established such that the postconditions of a preceding step will assert propositions which satisfy the preconditions of the following step, and the step at the top of this hierarchy asserts a proposition which satisfies the initial goal. This hierarchy imposes a total order on steps within one temporal chain, but only a partial order across the full step set. Since the preconditions of the step which asserts the initial goal are achieved in the same way as the initial goal, these preconditions (or the steps themselves) are called subgoals of the initial goal.

We have adopted this representation for GSG with some modifications. Preventer goals are most like the original sets of subgoals in plan representations. We have added subgoal sequences to somewhat collapse the deep hierarchy that results from temporal chains, and to allow a straightforward way to selectively activate only the goals which are currently being acted upon. The run-time information environment of our system does not need access to preconditions and postconditions of steps to generate step ordering information. Instead, it needs a set of co-conditions to monitor those preconditions which must remain satisfied until an action's postconditions have been achieved. This interpretation of preconditions matches well with the condition expectations we identified in our initial domain explorations. These have become our requisite goals. Also, instead of asserting postconditions, our system needs to monitor the real process and recognize when the postconditions that an action was intended to produce have been accomplished. Postconditions have been replaced with the success-criterion condition, and a problem-yet condition has been added to monitor the system and ensure that this happens in a timely fashion. Thus, in GSG, event expectations have taken the form of expectations about goal satisfaction.

Another approach, called Goal Tree/Success Tree modeling has recently been presented in [Modarres, *et al.*, 1985] and [Kim and Modarres, 1986]. It uses goal representations very similar to typical planning representation to encode information about goals for continuous processes and hierarchies of equipment combinations to provide real time advice to nuclear power plant operators.

We believe that GSG is capable of representing any batch or continuous process which consistently follows a standard operating procedure. All batch manufacturing processes are analogous to continuous processes during the completion of individual phases, and all continuous processes have a batch component to them in start-up and shut-down operations. The requirement for a standard operating procedure must be imposed since this system has no planning capabilities of its own.

We have considered adding a planning component to COOKER. Occasionally a problem will occur in the plant which undoes an affect which had been achieved by an earlier goal, and the plant needs to go through the process of re-satisfying that earlier goal. A planning component could be made which could schedule the earlier goal to be satisfied again. However, we have found that the plant engineers with whom we have worked have extensive standard operating procedures for handling these situations, and they do not need nor want our system to synthesize novel problem resolution strategies on the fly.

We have also found that GSG does not represent diagnostic procedures in a clean way. Operators will occasionally violate their usual condition expectations in order to test a component, and GSG identifies these violations as problems. Also, we are looking at a different representation for requisites since, for example, it makes little sense to have a goal with a sequence as a requisite. At Honeywell we are continuing to explore GSG and other representations for manufacturing processes.

IV. Conclusion

The GSG system which we have described here is somewhat simpler than the current implementation. Since our initial design, we've added support for three-valued logic, lattice interconnections between goals, conditional goal activation, automatic goal synchronization with processes in progress, assumption fields, an incremental, dynamic problem solving mechanism, and we've defined a GSG Language which is translated into Zetalisp code through a goal compiler. Despite these further developments, the basic representation and methodology has remained constant. All process information is stored in goal objects which traverse their sequences of subgoals, and monitor their conditions. This approach has proven to be computationally efficient, taking an average of 477 milliseconds (elapsed time with the operator interface and data gatherer running) for top level loop SATISFIEDP processing across a set of goal lattices totaling approximately 800 goals.

The GSG representation has worked well, allowing us to implement and quickly install an initial knowledge base which could recognize problems in all phases of the plant's process, and then later add to that knowledge base to diagnose more problems and give more detailed advice about particular problems. We've been able to reuse many portions of our initial goal trees within subsequently developed branches, speeding up the process of encoding detail about further phases. The approach of positively representing what should happen within the process has allowed us to use a single representation to serve the two functions of process monitoring and problem diagnosis.

Acknowledgements

We'd like to recognize the rest of the COOKER project team for their contributions to this work. They are Emilio Bertolotti, Arch Butler, Paul Christopherson, Kim Hermanson, Anne M. Hossfeld, Ron Mawby, John Nomura, John Sederberg, and Alan Wolff. Thanks to Camille Bodley, Steve Harp, Anne M. Hossfeld, Ron Joy, Carol Kaemmerer, Kurt Krebsbach, John Nomura, Jim Richardson, and Alan Wolff for their comments on this report.

References

[Chapman, 1985] D. Chapman, *Planning for Conjunctive Goals.* Technical Report AI-TR-802, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

[Kaemmerer and Mawby, 1986] W. F. Kaemmerer and R. Mawby, Representing Knowledge About Expectations in a Real-time Expert Advisor for Process Control. In *Proceedings ISA-86,* pages 809-820, Houston, Texas, Instrument Society of America International Conference, October, 1986.

[Kaemmerer and Allard, 1986] W. F. Kaemmerer, J. R. Allard, An Automated Reasoning Technique for Providing Moment-by-Moment Advice Concerning the Operation of a Process. In *Proceedings AAAI-87,* Seattle, Washington, American Association for Artificial Intelligence, July, 1987.

[Kim and Modarres, 1986] S. Kim, M. Modarres, Application of Goal Tree-Success Tree Model as the Knowledge-Base of Operator Advisory Systems, Submitted for Publication to *Nuclear Engineering and Design Journal*, October 1986. Correspondence to M. Modarres, Department of Chemical and Nuclear Engineering, University of Maryland, College Park, MD 20742.

[Modarres, *et al.*, 1985] M. Modarres, M. L. Roush, R. N. Hunt, Application of Goal Trees in Reliability Allocation for Systems and Components of Nuclear Power Plants, *Proceedings of the Twelfth International Reliability Availability Maintainability Conference for the Electric Power Industry*, Baltimore, MD, April, 1985.