CONF-831047--7

AUTOMATED REASONING IN MAN-MACHINE CONTROL SYSTEMS*

CONF-831047--7

DE83 014307

by

R. C. Stratton

E. L. Lusk

EBR-II Project

Argonne National Laboratory

P. O. Box 2528

Idaho Falls, Idaho 83401

Submitted for Publication
in Nuclear Safety

*Work supported by the U. S. Department of Energy under Contract W-31-109-38.

[illegible]

AUTOMATED REASONING IN MAN-MACHINE CONTROL SYSTEMS*

by

R. C. Stratton

E. L. Lusk

EBR-II Project

Argonne National Laboratory

P. O. Box 2528

Idaho Falls, Idaho 83401

Submitted for Presentation
at the
American Nuclear Society
1983 Winter Meeting
October 30-November 4, 1983
San Francisco, CA

*Work supported by the U. S. Department of Energy under Contract W-31-109-38.

Automated Reasoning in Man-Machine Control Systems

Ewing L. Lusk

Mathematics and Computer Science Division
Argonne National Laboratory

Rex Stratton

Experimental Breeder Reactor II Division
Argonne National Laboratory

ABSTRACT

This paper describes a project being undertaken at Argonne National Laboratory to demonstrate the usefulness of automated reasoning techniques in the implementation of a man-machine control system being designed at the EBR-II nuclear power plant. We show how automated reasoning influences the choice of optimal roles for both man and machine in the system control process, both for normal and off-normal operation. In addition, the requirements imposed by such a system for a rigorously formal specification of operating states, subsystem states, and transition procedures have a useful impact on the analysis phase. Finally we discuss the definitions and rules for a prototype system which is physically simple yet illustrates the some of the complexities inherent in real systems.

1. Introduction

Control mechanisms for complex physical systems consist of men and machines. The machines may include data acquisition devices, completely automatic control equipment, and plant computers for data analysis, semi-automatic control functions, and communication with a human operator. Both men and machines are information processing systems[13]. Argonne National Laboratory's Man-Machine Control System takes the point of view that each should be viewed as a component of the control system. Design of the control system thus involves allocation of various monitoring, diagnosis, and control functions to those controller subsystems which can best perform them. Recent advances in automated reasoning technology[16] make possible greater optimization of both man and machine controller components by reallocating some of the above-mentioned functions to the machine component of the control system.

In order to effectively control a physical system the controller requires a model of the system and data about the states of various components and subsystems. Also needed are rules for deducing the present and allowable future states of the physical system from the system model and the data. (To be more exact, the deductions lead to a conclusion about the state of the model rather than of the system itself. How closely this approximates the state of the physical system depends on how precisely the rules are formulated and applied.) Finally, procedures are needed for effecting transitions among states of the physical system.

In classical procedural control most of the necessary deduction function is performed by the human component of the control system. The operator maintains a model of the system based on his training and experience. Presented by the machine component with data about the states of various system components, he uses his model to deduce the overall state of the physical system. Aided by an extensive set of rules embodied in a set of procedure manuals, he deduces what control actions should be performed. Encountering a situation not covered by predefined procedures, he relies on rules just the same. These are rules he has absorbed as part of his training and operating experience, and may have varying degrees of precision and applicability. (The precise name for a rule with imprecise applicability is "rule of thumb.") Such rules are indispensable in rapid diagnosis of problems and identification of necessary control procedures.

It is desirable to make the deduction function of the controller reliable, predictable, and complete. One way to do this is to formulate rules for state diagnosis and procedure prescription in a form understood by an automated reasoning system. Moving parts of the deduction function from the human to the machine component of the control system frees the operator for other functions for which he is optimally suited. Other advantages of this approach become apparent in the analysis phase of system implementation, as described below. Work has been done in this area, and in some cases is still ongoing, at Westinghouse Hanford[14], E.G.&G[11], Combustion Engineering and System Control Incorporated[10], and Institut für Atomenergi and Gesellschaft für Reactorsicherheit[1].

In this paper we describe a project which has been formulated to explore and demonstrate the advantages of this approach. The reasoning system used is the Logic Machine Architecture (LMA) system developed at Argonne National Laboratory[5, 6, 7]. It in turn is an outgrowth of a long-running research project in automated deduction[2, 3, 8, 9]. The system is being integrated into the man-machine control system being designed for a subsystem of an experimental breeder reactor operated by Argonne in Idaho (EBR-II[15]). We describe here a fictitious prototype system which is meant to illustrate the use of automated deduction techniques; the real system model to which the techniques will be applied is still in the definition phase.

Formal definition of the prototype system and experimentation with its operation has demonstrated that the reasoning techniques encapsulated in LMA are directly applicable to the design of man-machine control systems. They can be used as an analysis tool to formulate a system model, as a diagnostic tool to determine system states, and as an information system to contain the system rules.

2. The Reasoning Subsystem

In this section we describe more fully the reasoning subcomponent of the control system.

2.1. System Architecture

One function of the Man-Machine Control System is the deductive function. This function will be at least partially performed by the automated reasoning subsystem, which will be a software subfunction of the MMCS. The reasoning subsystem will be general-purpose one, driven by a knowledge database containing the encoding of the system model and the rules which govern the operation of the model. The model and rules can be easily changed by modifying the contents of this database. Many expert systems are implemented as special-purpose, stand-alone systems, but we believe that a gereral-purpose reasoning system can be even more useful, provided that it is implemented as a package of reasoning tools which can be called upon as needed by a higher level, special-purpose system. Additional functions to be provided by the higher-level system would include interfacing to special hardware for data acquisition, interaction with a human operator, generation of certain automatic control signals, etc. These can all be performed independently of the reasoning subsystem.

Fortunately, a reasoning system with the appropriate architecture has recently been implemented. It is Logic Machine Architecture[5, 6], which we will refer to throughout as LMA. LMA is a package of subroutines which encapsulate functions required to perform reasoning. It is the latest implementation of the ideas developed over the last twelve years by the Argonne National Laboratory-Northern Illinois University research project in automated reasoning.

2.2. Structure of the Reasoning Subsystem

The architecture of the reasoning subsystem is described in detail in [5] and [6]. We give here a brief overview.

LMA is comprised of approximately fifty thousand lines of code, and is written in Pascal. It was designed to be extremely portable, and has been ported from its development environment, VAX/UNIX, to such other environments as VAX/VMS, IBM/CMS, Apollo, and Perq. Moving it to a new environment is quite straightforward, and can ordinarily be done in a few days.

LMA is divided into software layers. Procedures residing in one layer have access only to those procedures residing in the next lower layer. Thus one can experiment with alternate implementations of certain algorithms without impact on the rest of the system. The highly modular structure increases reliability and maintainability.

The lowest layer (Layer 0), which is the one layer accessible to all higher layers, implements certain abstract data types not provided by the host language. Some of these are indefinitely extendible character strings and vectors of integers. Such data types free the higher layers from dependencies on limits to the size or complexity of problems considered by the system.

Layer 1 contains primarily the database support functions required to manage large knowledge bases of logical formulas. It allows rapid access to the relevant facts which meet certain requirements imposed by the higher layers.

Layer 2 provides the inference mechanisms themselves: procedures which implement a wide variety of clause-based inference rules, subsumption, simplification and canonicalization of formulas, etc. The procedures which make up Layer 2 provide the most general yet complete set of reasoning tools. Descriptions of the principal features implemented can be found in[16]. Above this level, multiple reasoning systems are being implemented.

At Layer 3, an interactive clause-based theorem prover has been completed[4]. It has provided the environment for experimentation which has resulted in the representation of plant processes described in this paper. It is anticipated that a special-purpose layer 3 reasoning program will be required to interface the LMA-based reasoning component to the overall Man-Machine Control System. Such a layer 3 mechanism is now being designed.

3. The Prototype System

3.1. System Description

The prototype system chosen for experimentation and demonstration purposes is shown in Figure 1. This system was constructed to provide some aspects of a real system while limiting the degree of complexity for tutorial purposes. The function being modeled is a temperature control function. The physical components supporting this function (in this particular abstract model) are a parallel pair of electromagnetic pumps in the secondary loop, controlled by one of two parallel automatic flow controllers. In addition, we postulate the operator as a third potential flow controller, with the idea that in the event of failure of both automatic flow controllers, the operator can manually perform the flow control function by reading the output of the temperature sensor and manually controlling the secondary loop pumps. For the purposes of exploring this model we will assume that the behavior of the reactor itself is normal at all times. We will not include the heat transfer functions of the model (the reactor, the pump in the primary loop, and the heat exchangers.

The objective of the system, during reactor power operations, is to keep reactor inlet temperature (Tin) constant. This objective is achieved by varying the rate of flow, via the pumping function, of the heat transfer media in the secondary loop. The temperature control function senses Tin and adjusts the electrical current to the electromagnetic pumps as necessary. The variation in the pump current creates an increase or decrease in the media flow rate. This change in the media flow varies the system heat transfer capability and therefore the value of Tin.

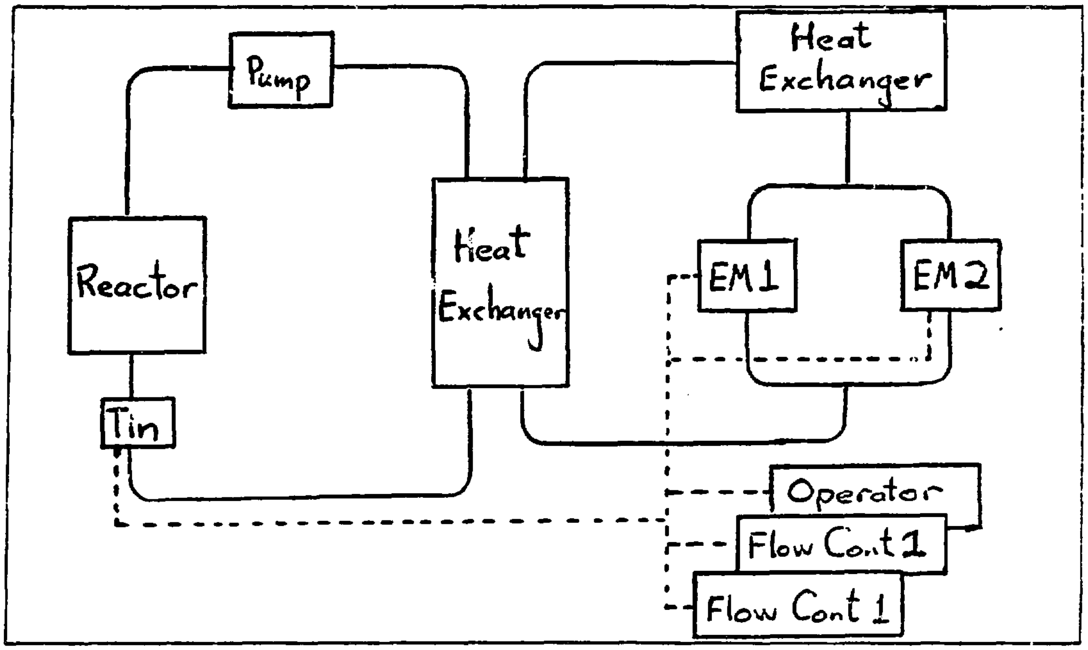

Figure 1.

3.2. Illustrative Features of the Prototype

Simple as it is, this model captures several features which will be found in a more realistic model of an operating plant.

The first is that there is a natural decomposition of the system into subsystems. We treat the parallel pair of pumps in the secondary loop as a subsystem, as well as the control function provided by the two parallel flow controllers and the operator. The notion of a subsystem composed of parallel components, in which only one component out of a set is needed to perform the subsystem's function, is a typical one. The primary reason that the notion of subsystem is needed in a real system, however, is to deal in a coherent manner with the complexity provided by a large number of individual components.

We will allow our prototype system three normal system states: On (operating), Stdby (startup), and Off (shutdown). Appropriate responses to events will depend on both actual and intended states of the system, as would be true of a real, more complex model.

The prototype contains elements of varying levels. That is, immediate subsystems of the system include both indivisible components like the primary loop temperature sensor and subsystems like the secondary pumping system, which has an internal structure. We will not distinguish among high-level systems, subsystems, and individual components, referring to all of them as *elements*. This allows a degree of uniformity in our rules which describe the system which would otherwise not be possible, and also allows us to change a component into a subsystem with minimal impact on other rules. (An example of this change might be the division of the electromagnetic pumps into their mechanical and electronic subcomponents.) The allowable states for each element are given in Table

1.

Allowable System, Subsystem, and Component States						
Element	On	Off	Stdby	Maint	Isolated	Failed
System	x	x	x	x	x	x
TempSensing	x	x		x	x	x
FlowControl	x	x	x	x	x	x
Pumping	x	x		x	x	x
Tin	x	x		x	x	x
EM1	x	x		x	x	x
EM2	x	x		x	x	x
Controller1	x	x	x	x	x	x
Controller2	x	x	x	x	x	x
Operator	x	x	x			

Table 1.

4. A Short Introduction to Automated Reasoning

In this section we describe informally the language used to make statements about the system and the principal mechanism for deducing new statements from existing ones.

4.1. Notation

We employ a standard functional notation for assertions. Thus

State(EM1,Failed)

means that the state of element EM1 is "Failed." Compound statements can be made from these assertions, for example,

if State(EM1,Failed) then State(SecPumpSys,Inoperable).

Compound conditions are allowed:

if State(EM1,Operating) & State(EM2,Operable)
then State(SecPumpSys,Operating);

We impose the following restriction on compound statements. We allow only "and" connectives (abbreviated by "&") in the "if" part of compound statements and only "or" connectives (abbreviated by "|") in the "then" part. This means that each statement is a *clause*, which is the type of statement processed by the inference mechanisms of LMA. The restriction is not a serious one in the sense that "if-then" statements which are not clauses can be converted to one or more clauses without losing their meaning.

A further characteristic of clauses is that when variables appear, they are assumed to be universally quantified. This means, for example, that the statement if State(x,Maint) then State(x,Inoperable)

means that for any element x, if x is in the maintenance state, then x is inoperable.

4.2. Resolution-based Deduction

LMA provides a variety of rules for inferring new clauses from existing ones, thus expanding the knowledge base. The rule relied on most heavily is called *hyperresolution*. It corresponds to the normal human reasoning operation of inferring the conclusion of an "if-then" statement when all of the hypotheses are satisfied. It does not produce intermediate results when only some of the hypotheses are satisfied. For example, from the five clauses

1. if Need(SecPumpSys,On) & State(x,Operable) & Paired(x,y) & State(y,Operable) then Need(y,On)
2. Need(SecPumpSys,On)
3. State(EM1,Inoperable)
4. Paired(EM1,EM2)
5. State(EM2,Operable)

the system derives, using hyperresolution,

6. Need(EM2,On).

Note that partial deductions are in theory possible from subsets of the above set of clauses. For example, from 1, 2, 3, and 4, we could legitimately infer

if State(EM2,Operable) then Need(EM2,On).

Such deductions are made by some resolution-based inference rules. In the interest of efficiency and because they are not necessary, we block derivation of such results by using hyperresolution.

4.3. Relationship of LMA to Other Types of Reasoning Systems

Various types of systems have appeared to make it easier for end users to apply automated reasoning techniques to their specific problems. In this section we briefly explain the relationship of the mechanism just described (resolution) to two other families of systems, namely "logic programming" and "expert systems."

Logic Programming (of which the PROLOG system is the best-known) is quite similar to the resolution approach, but with some very specific and important differences. It starts with a collection of facts expressed as clauses, with the restriction that compound conclusions in "if-then" statements are not allowed. This initial collection of facts is not expanded, as in the resolution approach. Rather, specific queries are submitted to the system, and the system determines whether the answer can be deduced from the available information. Thus the reasoning is more algorithmic, being directed toward a specific goal. It is anticipated that some of the deductions required in the Man-Machine Control System can be made more efficiently using this approach, and a PROLOG-like subcomponent is being added to LMA.

Expert Systems are computer programs which provide "expert" advice to humans engaged in some activity. In that sense, any procedure prompting system, including the one being described here, is an expert system. The term as most often used also identifies a particular style of program characterized by a sophisticated user interface, a large number of relatively specific rules (similar to "if-then" clauses), and a simple reasoning component. Sometimes the reasoning component is capable of drawing probabilistic conclusions when making a diagnosis.

LMA corresponds to the reasoning component of an expert system, but is much more powerful than the reasoning components typically employed in expert systems. It does not contain a general-purpose user interface. For experimentation and demonstration purposes, intthp[4] provides such an interface. In the project under discussion, a special-purpose interface to the reactor operator will be provided by other components of the MMCS.

5. The Prototype Knowledge Base

In order to control an element the controller requires a model of the element, rules that describe the relationships among the states of an element and the states of its subelements and how to effect changes of state, and finally facts about the current state of the element. The clauses which the automated reasoning subsystem of the MMCS operates fall into three categories: ***facts***, ***model structure***, and ***rules***.

5.1. Facts

The basic facts which the system reasons from are the states of the various components in the system. The automated reasoning subsystem relies on the rest of the MMCS for these facts, and is not concerned with the sources of such information. The facts come from the plant administrative and parametrical status. Some of these facts come from the Component Isolation Control System (CICS) currently being developed at EBR-II; others from the plant Data Acquisition System. Still other facts may come from operator input, such as the desired operating mode of the system. The reasoning system may even deduce that it must ask the operator for a specific item of information. In each case, it is the function of the special-purpose interface to translate the incoming information into clause form. Typically such clauses will be simple assertions, rather than "if-then" statements, but there is no restriction on clauses obtained in this way.

5.2. Model Structure and Rules

The most demanding aspect of the project is to formulate the clauses which correspond to rules for deducing the states of elements from their subelements, for deducing the correct sequence of operations necessary to achieve a desired state, and choosing new goals when original ones cannot be reached. It is justified by the end result of having a flexible and easily modifiable representation of knowledge about how to operate the plant.

Each element of the system will be able to assume a finite number of different states and the element's behavior will be described and controlled by unambiguous transitions and inter-element relation rules covering all transitions between these states as affected by the states of the element and its subelements[12]. To develop transition rules, element structure and state definition must be established. The element model provides the structure and definition of the system. The states define the allowable configuration of the system elements and their interrelationships. Rules are then defined using the model and the states. The model structure and the rules are naturally integrated in the clause formulation we describe here.

In this section we describe briefly the families of rules which were developed in the course of studying the prototype system. As in the case of the design of the prototype system itself, we have made certain assumptions about the prototype system whose primary purpose is to illustrate how certain realistic configurations might be represented without introducing unnecessary complexity. We do not give here the complete set of rules, but indicate the sorts of rules used.

5.2.1. Rules about Families of States

It is useful to collect some of the possible states into groups, representing certain more abstract states. In many cases rules do not need to specify one of the basic states for an element, but rather one of these abstract states. This led to a collection of clauses of the following type.

```
if State(x,On) then State(x,Operating)
if State(x,Off) then State(x,Operable)
if State(x,Stdby) then State(x,Operable)
if State(x,Iso) then State(x,Operable)
if State(x,Maint) then State(x,Inoperable)
if State(x,Failed) then State(x,Inoperable)
if State(x,Operable) then State(x,NotOperating)
if State(x,Inoperable) then State(x,NotOperating)
```

Note that this structure makes it relatively easy to add new states if experimentation shows they will be useful. Rules which utilize the more abstract states need not be changed.

5.2.2. Rules about Subsystem States

Another set of rules govern the deduction of the state of a complex element (one with subelements) from the states of its subelements. Some of these are:

```
if State(Flow1,On) & State(Flow2,Notoperating) &
  State(Man,Operable) then State(SecFlowSys,On)
if State(Tin,On) & State(SecPumpSys,Operable) &
  State(SecFlowSys,Operable) then State(System,Startup)
```

Again note that stating the requirements for subsystem states in clause form rather than embedding them in the code of a special-purpose program makes it easy to modify such definitions.

The "Need" predicate was introduced to designate requirements. Thus

```
    Need(EM1,On)
```

means that a condition has arisen that requires that EM1 be in the On state. In order to express the ability or ability to respond to such a requirement, the "Cando" and "Cantdo" predicates are introduced. This leads to clauses such as

```
    if Need(x,On) & State(x,Inoperable) then Cantdo(x,On)
    if Need(x,On) & State(x,Operable) then Cando(x,On).
```

The "Paired" predicate is used to express the fact that a set of components operates in parallel, with only one required to perform the function of the sub-system they comprise. Thus rules about such components can be stated in a general form. In our prototype system, we have

```
    Paired(EM1,EM2)
    Paired(EM2,EM1)
    if Need(SecPumpSys,On) & State(x,Inoperable) & Paired(x,y)
      then Need(y,On).
```

When it is deduced that a component can be put in a desired state and operator action is required to do so, communication with the operator must take place. We represent this with the special predicate "$OUT," whose implementation can be customized. A typical use might be

```
    if Need(Pump,On) & Cando(Pump,On) then $OUT("Turn Pump on").
```

If the clause

```
    $OUT("Turn Pump On")
```

is deduced, the message will be conveyed to the operator.

5.2.3. Rules about State Sequences

The final collection of rules describe the sequence of states to be attempted, in the case it is discovered that a desired state cannot be achieved. It contains such rules as:

```
    if Need(System,Operating) & Cantdo(System,Operating)
       then Need(System,Startup).
    if Need(System,Startup) & Cantdo(System,Startup)
       then Need(System,Shutdown).
```

We can include the rules for notifying the operator of deductions make about changes of goal by using the $OUT predicate with variables. When a list of values is to be communicated to the operator, $OUT uses the LMA representation of lists, using the C (for concatenation) function. The list containing a, b, and c is represented by C(a,C(b,C(c,NIL))). Thus we have clauses like

```
    if Need(x,y) & Cantdo(x,y)
      then $OUT(C("Cannot put",C(x,C("in",C(y,C("state",NIL))))))
```

6. Conclusion

There are two principal conclusions that can be drawn from this developmental work. Automated reasoning can play a significant role in the Man-Machine Control System and can be effectively utilized in the design phase of the machine part of the control system.

Automated reasoning, when applied to the Man-Machine Control System, optimizes the man and machine roles by absorbing part of the deduction function traditionally performed by the human subcomponent of such a system. The reasoning system integrates the system facts with the model structure and rules, and provides the automated output of a system control response to be executed by some other component of the MMCS.

The use of an automated reasoning system is an effective analysis tool that can be utilized during the design phase of electromechanical control systems to enhance and trouble-shoot the design logic. The formal language used by the reasoning system forces rigor in the rules and model definition of the control system. Also, subsequent to the system implementation the designer can interactively operate and manipulate the control system to determine if any unforeseen normal or off-normal operating conditions are unaccounted for in the design. It should be noted that the above can be done prior to any hardware development.

References

1. W. Bastl and L. Felkel, "Star, A Disturbance Analysis System and its Application to a PWR-Station," pp. 285-292 in *Conf-800403*, (1980).
2. E. Lusk and R. Overbeek, "Data structures and control architecture for the implementation of theorem-proving programs," in *Proceedings of the Fifth Conference on Automated Deduction, Springer-Verlag Lecture Notes in Computer Science, v. 87*, ed. Robert Kowalski and Wolfgang Bibel, (1980).
3. E. Lusk and R. Overbeek, "Experiments with resolution-based theorem-proving algorithms," *Computers and Mathematics with Applications* 8(3) pp. 141-152 (1982).
4. E. Lusk and R. Overbeek, *An LMA-Based Theorem Prover*, (preprint). November 1982.
5. E. Lusk, William McCune, and R. Overbeek, "Logic machine architecture: inference mechanisms," pp. 85-108 in *Proceedings of the Sixth Conference on Automated Deduction, Springer-Verlag Lecture Notes in Computer Science, v. 138*, ed. D. W. Loveland, (1982).
6. E. Lusk, William McCune, and R. Overbeek, "Logic machine architecture: kernel functions," pp. 70-84 in *Proceedings of the Sixth Conference on Automated Deduction, Springer-Verlag Lecture Notes in Computer Science, v. 138*, ed. D. W. Loveland, (1982).

7. E. Lusk and R. Overbeek, *Logic Machine Architecture Inference Mechanisms - Layer 2 User Reference Manual*, (preprint). 1982.

8. J. McCharen, R. Overbeek, and L. Wos, "Complexity and related enhancements for automated theorem-proving programs," *Computers and Mathematics with Applications* **2** pp. 1-16 (1976).

9. J. McCharen, R. Overbeek, and L. Wos, "Problems and experiments for and with automated theorem-proving programs," *IEEE Transactions on Computers* **C-25**(8) pp. 773-782 (1976).

10. C. H. Meijer, B. Frogner, and A. B. Lang, "A Disturbance Analysis System for On-Line Power Plant Surveillance and Diagnosis," in *IAEA/NPPC Specialists Meeting on Procedures and Systems for Assisting an Operator During Normal and Anomalous Nuclear Power Plant Operation Situations (GRS-19)*, ().

11. W. R. Nelson, "Reactor: An Expert System for Diagnosis and Treatment of Nuclear Reactor Accidents," *EGG-M-09782*, (1982).

12. John von Neumann, *The Theory of Self-Reproducing Automata*, University of Illinois Press (1966).

13. J. Rasmussen, "The Human as a System Component," pp. 68 in *Human Interaction with Computers*, (1980).

14. E. Seeman, R. W. Colley, and R. C. Stratton, "Optimization of the Man-Machine Interface for LMFBR's," *HEDL-SA-2727*, (1980).

15. R. G. Staker and D. W. Cissel, "EBR-II: More Than a Decade of Experience with an LMFBR Power Plant," in *Proceedings of the American Power Conference*, (1976).

16. L. Wos, S. Winker, and E. Lusk, "An automated reasoning system," *Proceedings of the AFIPS National Computer Conference*, pp. 697-702 (1981).

Appendix A
A Sample Session

In this appendix we show some output from the prototype system, still utilizing the interface provided by intthp.

In the first example, states of components are entered which correspond to startup mode for the system. The operator specifies that operating mode is desired. The reasoning system correctly deduces the current mode of the system and provides instructions on how to reach operating mode. System responses are given in bold face; user input is given in normal type.

```
Modes of Plant are Operating, Startup, Shutdown
Enter desired mode of operation
> Operating;

States of components are On, Off, Iso, Stdby, Failed, Maint
Enter state of EM1
> Off;
Enter state of EM2
> Off;
Enter state of Tin
> On;
Enter state of Flow1
> Off;
Enter state of Flow2
> Off;
Enter state of Man
> Stdby;

System is in Startup mode

Turn EM1 on
Turn Flow1 on
```

In the second example, we enter states of components corresponding to a situation in which the system was in operating mode when pump EM1 failed. The automated reasoning system deduces that the system is in an off-normal mode, and recommends action to place it back into the desired mode.

```
Modes of Plant are Operating, Startup, Shutdown
Enter desired mode of operation
> Operating;

States of components are On, Off, Iso, Stdby, Failed, Maint
Enter state of EM1
> Failed;
Enter state of EM2
> Off;
Enter state of Tin
> On;
Enter state of Flow1
> On;
Enter state of Flow2
> Off;
Enter state of Man
> Stdby;

System is in off-normal mode

Turn EM2 on
```