To appear in *Computer Integrated Manufacturing in the Process Industries,* April 1994.

Formal Specification and Verification of Real-Time Sequential Control Systems*

Armen Gabrielian

UniView Systems
1192 Elena Privada
Mountain View, CA 94040
USA

ABSTRACT

A variety of methods have been employed for specifying the behavior of sequential control systems that, e.g., can be implemented by programmable logic controllers (PLC's). In dealing with real-time issues, timed Petri nets and other popular formalisms provide limited capabilities with respect to both specification and analysis of behavior. At the same time, many of the *formal methods* that have been reported in the literature are not readily applicable to the area of sequential control. In this paper, we explore the applicatior of a formal method for real-time systems based on "hierarchical multi-state (HMS) machines," to the specification and verification of sequential control systems. Verification in this context goes beyond simulation and testing by attempting to provide mathematical proofs for correctness of behavior with respect to general "safety properties." HMS machines, which were originally introduced in 1988, consist of parallel and hierarchical automata in which multiple states are true and multiple transitions can fire simultaneously. The temporal behavior of an HMS machine is defined in terms of an interval-based temporal logic. Examples of applications of HMS machines to sequential control problems and an overview of verification approaches are presented.

INTRODUCTION

The need for *formal methods* for specification and design verification of safety-critical systems has been recognized for a number years. With increased use of automation in many industrial applications, there is a significant opportunity for improving the state-of-the-art in process control design and development by using formal methods. By decreasing the risk of failure, formal methods can (1) reduce the danger to human lives and (2) minimize major potential financial losses. As indicated in Laprie and Littlewood [9], one can no longer rely on "engineering judgment" to assure safety in design. Also, as predicted in Chandy and Kesselman [1], in the future, "programmers, tired of debugging difficulties, will use design methods that produce correct programs as well as formal methods of proving correctness and methods of integrating proofs and testing." The most promising and important stage of design has been recognized to be the early phases of requirement specification and conceptual design. Errors in these stages are potentially the easiest to find, yet, if undetected, often turn out to be the hardest and costliest to correct at later stages.

* This work was sponsored in part by the Office of Naval Research under contract No. N00014-93-C0100.
APPROVED FOR PUBLIC RELEASE - DISTRIBUTION UNLIMITED.

Key characteristics of process control systems are concurrent occurrences of multiple events, the unexpected arrival of inputs, and the existence of temporal constraints on behavior. The inputs may consist of communication data in a network, sensor outputs in a chemical plant or vital signs of a patient in a hospital. The temporal constraints are now understood to be mostly *derived* requirements that emerge from physical considerations, laws of nature, limitations of computational resources or human factors considerations.

The consistent statement of requirements and the verification of design are critical in the development of process control systems. What is lacking is a language of communication between the customer of a process control system, who may not be an expert in process control, and the system designer. This language must be able to specify in a precise form what is currently stated informally in English or is merely communicated in oral form to the system designer. Besides the ability to specify a problem, two other features are extremely valuable: (1) the ability to execute a specification for experimentation before the actual creation of the process control application and (2) the capability for formally verifying critical design requirements.

Executability of a specification permits the simulation of behavior of a system and its environment. This can lead to elimination of errors that often arise in a statement of requirements for a complex system. Experimentation with an executable specification will also help eliminate unexpected side-effects of behavior.

While execution of a specification can help find most of the errors in a design, certain critical conditions require a stronger form of verification. Much research has been performed on formal verification of real-time systems in recent years that can have direct application in process control. In particular, given a formal specification of a process control system, one can attempt to verify its "safety properties," which are requirements that must *always* be satisfied. Symbolically, a safety property can be expressed in the language of temporal logic as $\Box p$, where p may be a conditional statement about the *history* of a system. For example, p may state that if a valve is open, the temperature must have been less than a pre-specified limit for the last 5 seconds. The concept of safety property is very general and encompasses most of the properties of interest in the design of process control systems and real-time systems in general.

Various approaches have been used in the past to describe the required behavior of sequential control systems that, e.g., can be implemented by programmable logic controllers (PLC's). Finite-state machines, variations of Petri nets, logic diagrams, and ladder diagrams are examples of notation systems that have been adopted widely for describing such systems. Finite-state machines and extensions have also been used for many years in a variety of other fields to describe the dynamic behavior of systems. For complex systems, however, the state explosion problem limits the usefulness of finite-state machine-based models. Also, it is difficult to model concurrency and time in terms of finite-state machines. Some of these problems are alleviated by using variation of Petri nets, in which concurrency can be modeled more easily. A version of Petri nets, called "grafcet" (David and Alla [2]) has become popular in Europe and a similar notation, called "sequential function charts" (Filer and Leinonen [3]), has been promoted in the United States. While numerous studies have been reported on analysis of Petri nets, except for simulation and exhaustive analysis, few practical approaches exist for analyzing the behavior of Petri nets *formally*. Analysis of timed-Petri nets is even more difficult. Finally, the implementation-oriented notation of ladder diagrams is too low level and complex to lend itself for formal analysis even for relative simple systems.

In this paper, we explore the application of the "hierarchical multi-state (HMS) machine" methodology (Gabrielian et al. [4,5,6,7,8]) to the specification and verification of *real-time* sequential control systems. We consider, in particular, processes with *discrete-valued* parameters that may control equipment and in which responses depend on the sequences of events in the past, as well as current conditions. The real-time aspect arises when, in addition to dependence on the *order of events*, temporal *constraints* are imposed on the behavior of a system. In the traditional approaches to modeling sequential control systems, the element of time is either not represented at all or is modeled in a very simplified fashion.

HMS machines provide a *formal* framework for specification of concurrent systems that (1) reduces the state space significantly compared to finite-state machines, (2) provides a simple automata-based execution semantics, (3) offers hierarchical decomposition capabilities, (4) employs an expressive graphic notation, and (5) uses a full temporal logic language for expressing complex temporal relationships. At the same time, one can often verify that a system specified in terms of an HMS machine satisfies a wide set of requirements, without exhaustive analysis of the state space.

This paper is organized as follows. The next section presents an overview of the main definitions. The following section offers two examples of English word descriptions that are transformed to formal specifications. We then review how previous work on verification can be applied to problems in the sequential control field. A brief summary appears at the end.

BASIC CONCEPTS AND DEFINITIONS

Intuitively, an HMS machine is a finite-state machine in which *multiple states may be true at a moment of time* and *multiple transitions can fire simultaneously*. In addition, an HMS machine state may hierarchically be decomposed into lower level HMS machines and the firing (or enablement) conditions of transitions are defined in terms of predicates in an interval-based temporal logic called TIL. We now expound briefly on each of these basic concepts.

Traditionally, a system has always been assumed to be in a single state at a moment of time. Mathematically, however, there is no fundamental reason for such a limitation. Two immediate advantages arise out of this generalization. First, N states can theoretically be used to represent a system that may require 2^N states in a standard finite-state machine. Secondly, as in Petri nets, one can model concurrency in a natural manner despite the use of the much simpler automata-theoretic behavior model. For example, unlike Petri nets, no token counting is employed in HMS machines and "interference" conditions are ignored.

Hierarchical decomposition has been studied for a variety of modeling techniques, including finite-state machines and Petri nets. In this regard, HMS machines are not significantly different. The main characteristics of hierarchies in HMS machines are (1) the use of a formal mechanism to distinguish the visible parts of a hierarchy from its invisible part, and (2) the localization of the effect of changes to the visible parts of a hierarchy only. The latter feature is very different from certain message-passing or broadcast schemes that are used widely in other representations. Our localized approach makes analysis much simpler, in general. Two other characteristics of hierarchies in HMS machines are (1) the use of different granularities of time at different levels of abstraction and (2) the availability of recursion in hierarchies (Gabrielian and Iyer [7]). We should note that recursive hierarchies are usually much more difficult to analyze. Nevertheless, we have been able to prove, e.g., certain correctness properties for unbounded first-in-first-out (FIFO) buffers represented in terms of recursive hierarchies.

The use of the temporal logic TIL to define firing conditions for transitions constitutes a major point of departure for HMS machines compared to other specification languages. In practically all previous approaches, time is modeled directly or indirectly in terms of *delays* on transitions. In HMS machines, a *relative notion of time* is employed such that, at each moment of time, one can determine whether a particular transition is enabled by evaluating an associated predicate in TIL. These TIL predicates use as parameters the *histories* of the states of an HMS machine. As a consequence, a much richer language is obtained for defining temporal constraints on behavior.

Another point of departure arises in the analysis of systems represented by HMS machines. While executing the underlying automaton model provides a means of simulating the behavior of an HMS machine, an alternative approach is obtained by using the TIL language to reason about history of states. In particular, since *any safety property* can be represented in terms of a state that becomes true if and only if the safety property is violated, one can reason about the possible sequences of events leading to a violation of a safety property without traversing the entire state space. Gabrielian and Iyer [8] describe a theorem proving approach based on such a technique.

Finally, nondeterminism in HMS machines is interpreted as follows: (1) transitions are explicitly designated to be either deterministic or nondeterministic, and (2) an enabled deterministic transition always fires, while a nondeterministic transition *may fire*. This increases the possible set of execution sequences but it simplifies analysis.

We now turn to the definition of the basic features of HMS machines. While we present some mathematical notation, the emphasis is on the intuitive understanding of the concepts. In the next section, we offer two examples along with the visual notation that will help clarify most of the definitions.

Definition 1. An HMS machine is a triple $H = (S, \Gamma_D, \Gamma_N)$, where

- S is a set of "states," any number of which may be true or "marked" at a moment of time
- Γ_D and Γ_N are, respectively, deterministic and nondeterministic sets of "transitions" of the form

  (PRIMARIES) (CONTROL) —> (CONSEQUENTS),

  where PRIMARIES $\subseteq$ S, CONSEQUENTS $\subseteq$ S and CONTROL is a "control predicate" or simply a "control" on the history of the markings of the states defined in an temporal interval logic called TIL. For a transition u, a state in its associated PRIMARIES (CONSEQUENT) set is called its "primary" ("consequent") state.
- A subset of S is denoted as its set of "ports"
- A state $s \in S$ may be an HMS machine H' itself. In that case, S is extended to include the port states of H'.

We now present an informal definition of the temporal interval logic TIL in which the control predicates for transitions are defined.

Definition 2. The temporal interval logic TIL is obtained by extending propositional logic with the following five operators:

$@(t)\varphi \Leftrightarrow \varphi$ true *at* t (denoted by $\mathbf{O}(t)$ in some previous papers)

$@^-(t)\varphi \Leftrightarrow \varphi$ true at *premoment* of time t (i.e., during a non-empty interval ending at t)

$[t,t']\varphi \Leftrightarrow \varphi$ true *always* from t to t′

$<t,t'>\varphi \Leftrightarrow \varphi$ true *sometime* from t to t′

$<t,t'>!\varphi \Leftrightarrow \varphi$ *became* true *sometime* from t to t′.

The current moment is considered to be time 0, with the past denoted by negative times. Thus, the TIL formula [-5,0]A is true if A has been true *continuously* from 5 moments ago to the present time, while the formula <-6,-2>A is true if A was true *sometime* from 5 moments ago to 2 moments ago (the unit of time is arbitrary and can be specified by the modeler). Intervals are assumed to be semi-closed (closed on the left and open on the right). Thus, $[t,t']\varphi$ may be true, while $@(t')\varphi$ is false. The premoment operator $@^-(t)$ is used only for technical purposes and is normally not necessary during specification.

Combined with the logical connectives of propositional logic (AND, OR, and NOT), one obtains a simple and powerful language to define both logical and temporal relationships among events and states in the description of the behavior of a system.

We now turn to the definition of the dynamic behavior of an HMS machine.

Definition 3. A transition γ in an HMS machine is "enabled" at time t, if for all $s \in$ PRIMS(γ), $@^-(t)s$, and @(t)CNTRL(γ).

Thus, a transition is enabled at time t if each of its primary states has been true for a non-zero semi-closed interval before t, and its control predicate is true at t. Forcing primary states to be true for a non-zero interval prevents anomalous behavior that may arise otherwise. This definition can be simplified if a discrete notion of time is adopted. In that case, a transition is enabled if for all $s \in$ PRIMS(γ), @(t-1)s, and @(t)CNTRL(γ).

Next, we present an informal definition of how the markings of states in an HMS machine changes over time.

Execution Rule. At each moment of time, all the enabled deterministic transitions and a subset of the nondeterministic transitions of an HMS machine "fire." If a transition u fires at time t, (1) all its consequents become true (or marked) at t, and (2) each primary state s of u becomes false or unmarked at t, unless a transition v fires simultaneously for which s is a consequent state.

We assume that initially a subset of the states of an HMS machine is true. Using the Execution Rule, we can then determine how the states evolve through time.

EXAMPLES OF SPECIFICATION OF SEQUENTIAL CONTROL SYSTEMS

Filer and Leinonen [3] present the following word description of a sequential control system example and its associated ladder diagram for implementation in terms of PLC's:

> **Textual Specification of a Sequential Lock:** Design a sequential lock that will operate a device for 15 seconds after three push buttons have been pressed in the correct sequence. If the incorrect sequence is pressed at any time, an error state should be activated that will prevent further tries for 10 seconds and then permit another try. If the error state is activated more than twice during a try, then the lock is to be disabled or locked out. A push button is to be provided to reset the lock if it is locked out.

In studying such word descriptions, one often encounters ambiguities, inconsistencies or missing elements. Even if the description is internally consistent, it often does not represent the requirements of the customer correctly. Studying a ladder diagram is not helpful because of its complexity and the fact that ladder diagram are at too low a level of abstraction. Figure 1 depicts the graphic representation of an HMS machine that reflects the desired behavior of the sequential lock indicated above. In this diagram, boxes denote states, thick arrows represent transitions and thin arrows indicate control conjuncts, with TIL predicates appearing next to the symbol Ⓣ. A two-level hierarchy is used in this specification, where the hierarchical state Error Handling encapsulates all the processes dealing with errors. The port states at the higher level (e.g., Reset) are controlled by the environment—in this case by the user. The "dependent states" are simply logical combinations of other states that are drawn separately to simplify the graphics.

In the HMS machine of Figure 1, we assume that Ready State is initially marked (as indicated by the small dark circle). The pressing of any out of sequence button leads to the Error Input port state of Error Handling. Once the Operate state is reached, a delay of 15 time units is necessary before the predicate [-15,0]Operate will become true, leading to the firing of the transition out of Operate and into Ready State. In this specific example, we have a very simple temporal condition that can be handled by a delay on the transition. It is easy to conceive of more complex temporal relationships that would require the creation of a number of *dummy states* if only delays are available for representing time. Such dummy states often make the understanding of behavior of Petri nets, for example, very difficult.

Within the Error Handling hierarchy, a potential ambiguity in the original word description can be noticed. The question is whether the system should be locked if two errors occur in *any* sequence of attempts or only if two errors occur before the Operate state is reached. Both interpretations are possible. In creating the formal specification in terms of an HMS machine, such ambiguities often become obvious and can be resolved through discussions with the customer. In this case, we have chosen to lock the system if two errors occur before the Operate state is reached. Thus, a record of a single error is deleted when the state Operate is reached.

Once an HMS machine representation for a system is created, its behavior can be analyzed through simulation. In addition, in the next section we will explore how one might attempt to verify formally that, e.g., the Operate state can never become true if the Locked state in Error Handling is true.

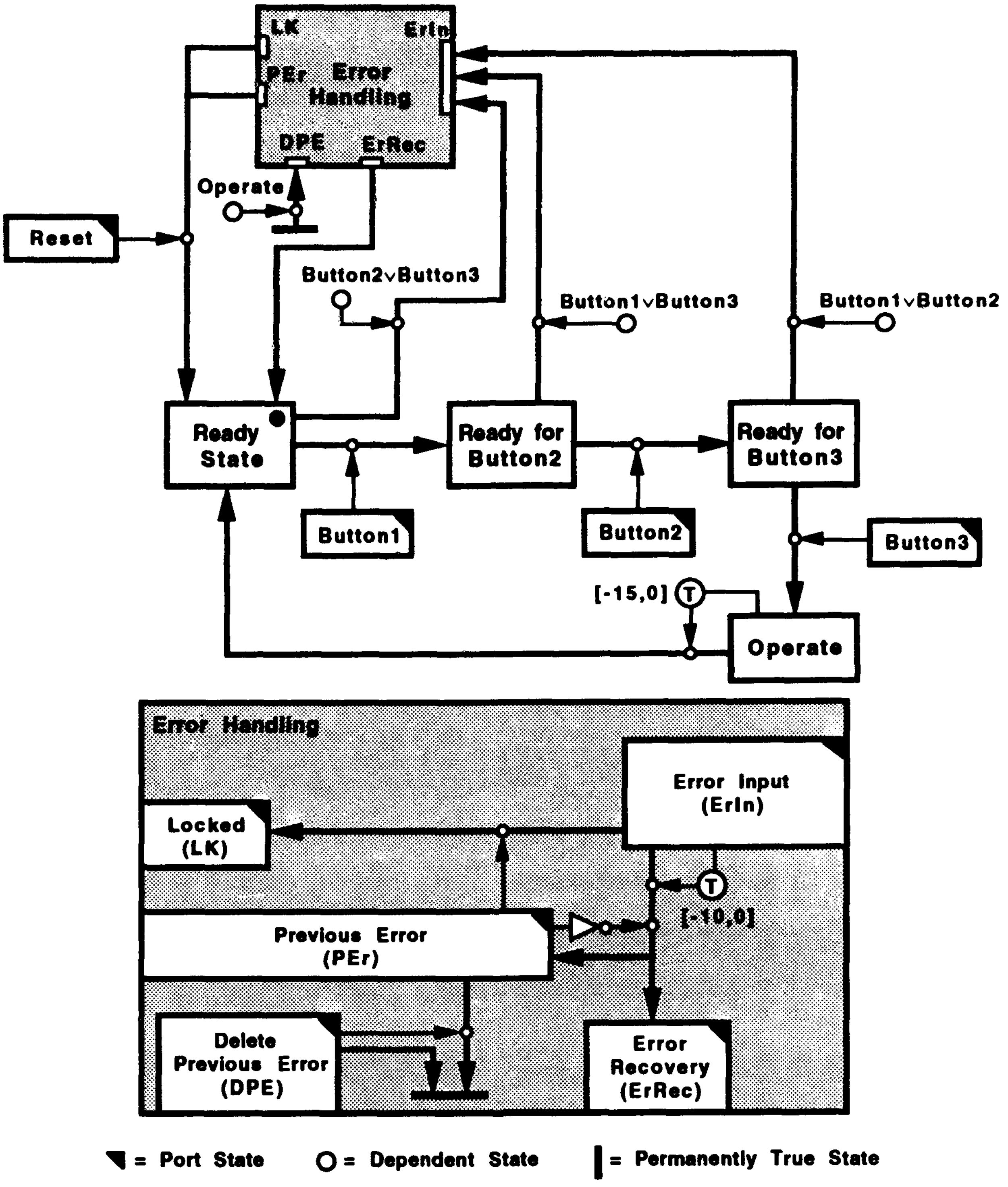


Figure 1. HMS Machine Specification of Sequential Lock

To consider another example, in Liptak and Venczel [10], the following word description of a sequential control systems is presented:

> **Standby Vacuum Pump.** The process must have high vacuum to proceed properly. Vacuum is normally maintained by an air ejector, but in case of failure or overload of the air ejector, the system pressure rises. The rise is sensed by a pressure switch (PSH), which automatically starts a vacuum pump, provided that a hand-actuated control switch (HS) for the pump motor is in the AUTOMATIC position. This switch also can be used to start and stop the pump manually. However, the pump is not permitted to start or run if the discharge temperature, as sensed by a temperature switch (TSH), is high or if the motor is overloaded and its circuit breaker is not manually reset. If high pressure is maintained for ten minutes, a high-pressure alarm (PAH) is actuated. High temperature is signaled by another alarm (IAH). If the pump control logic circuit loses power, the pump shall stop automatically but shall not be able to be restarted until the system is reset manually.

The standard logic diagram for this system also appears in Liptak and Venczel [10]. However, this logic diagram does not provide a convenient mechanism for reasoning formally about the dynamic behavior of the system. For example, it is difficult to detect hidden relationships that might exist among the Start, Automatic, and Stop manual controls.

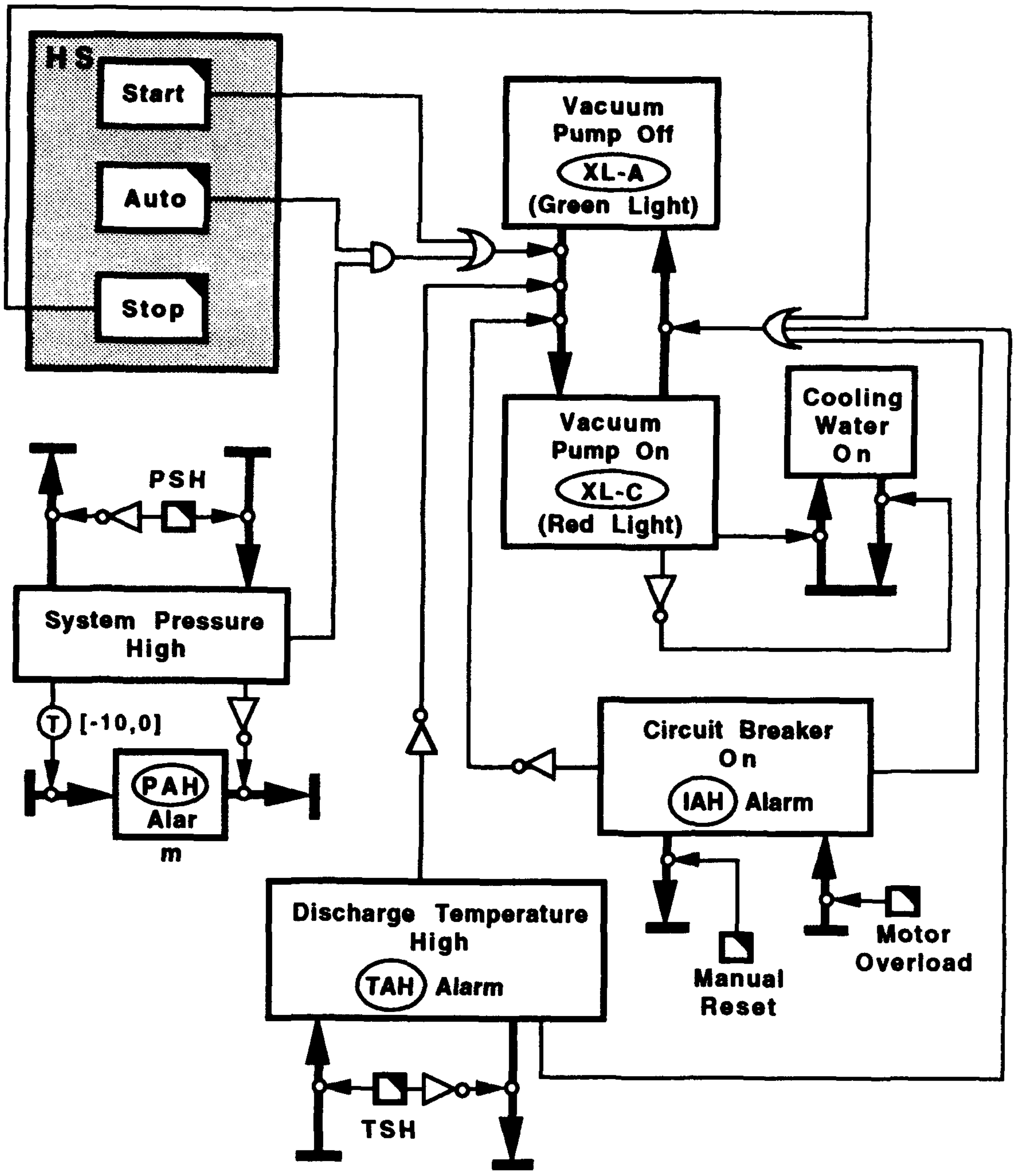
Figure 2 presents an HMS machine specification of the Standby Vacuum Pump. The graphic notation follows the conventions of Figure 1, except for the use of symbols for conjunction (AND) and disjunction (OR), which are adopted from standard VLSI notation.

The HS manual control switch in Figure 2 contains the three options of Start, Auto and Stop under a hierarchy. The lower-level details under this hierarchy are not provided here. If desired, it would be easy to define conditions that, for example, would permit only one of the states Start, Auto, or Stop, to be true at a moment of time.

Analysis of the HMS machine of Figure 2 suggests a possible omission in the original specification: no provision is made for the vacuum pump to stop operating when the system pressure is no longer high, even in the Auto state. This can easily be corrected by adding a control arrow with a negation from the state System Pressure High to the disjunctive predicate on the transition from Vacuum Pump On to Vacuum Pump Off in the upper right side of the figure.

In the diagram of Figure 2, the initial conditions are not indicated. Assuming that Vacuum Pump Off is true initially and Auto is chosen under HS, then the Vacuum Pump On state would become true if System Pressure High was true and Discharge Temperature and Circuit Breaker On were both false. Further analysis could be performed to determine the appropriateness of controls on other transitions.

For both of the examples in this section, two approaches can be adopted for the transition to implementation. First, one can create an implementation and develop a more detailed HMS specification for the implementation. To prove correctness, one can then show that the lower-level specifications are correct either independently or by demonstrating equivalence to the specifications presented here. Alternatively, one can define automatic methods of transforming formal specifications into an implementation language.



HS = Manual Pump Control

PSH = Pressure Switch

TSH = Temperature Switch

D = AND

) = OR

▷o = NOT

Figure 2. Specification of Standby Vacuum Pump

OVERVIEW OF APPROACHES TO VERIFICATION

A variety of techniques for verifying the properties of HMS machines have been developed. Simulation offers an experimental approach for confirming behavioral requirements. To obtain a greater degree of confidence in the correctness of a specification, more formal approaches can be undertaken. As mentioned earlier, in Gabrielian and Iyer [8] a "tableau-based theorem proving method" for verifying safety properties of HMS machines was presented. Recall that a safety property is a condition that must always be true for a system. Such a condition can relate to the *history* of the system, as well as its current state. Most properties of interest in real-time systems can be transformed into safety properties. To use this proof method, the following two steps are necessary:

- The safety property is represented in terms of a new "safety state" that becomes true if and only if the safety property is violated. It can be demonstrated that *any* safety property can be represented in this manner.

- One assumes that the safety property has become true and reasons *backward* to determine if such an assumption is possible. In the contrary case, one can demonstrate that all possible paths leading to the realization of the assumption lead to contradictions. Thus, a proof by contradiction can potentially be derived.

This theorem proving method is not completely automatic and requires human intervention. A prototype automated support tool is now available that provides assistance in performing tableau-based proofs and displays the proof steps graphically. Using this tool, very compact proofs have been generated for some well-known examples. An important extension of this method was recently obtained by introducing hierarchies into the theorem proving process. In this scheme, proofs are generated starting from the lowest level and are propagated upwards in a hierarchical fashion. Thus, at each level, a proof can be limited to *the current level of the hierarchy*. As such, this approach allows the integration of various proof methods, so that lower level proofs using completely different techniques can be incorporated into a tableau-based proof.

An alternate verification technique is obtained by using the scheduling analysis results of Gabrielian and Franklin [6]. In this approach, formal algorithms are used to generate mathematical requirements on *schedules* of concurrent event necessary to reach a goal state. By demonstrating the infeasibility of existence of schedules for reaching a safety state, one can then demonstrate that a system satisfies the associated safety property. A partial prototype implementation of this method has also been developed.

Enumeration techniques such as "model checking" can also be used to verify properties of sequential control systems represented in terms of HMS machines. Gabrielian and Iyer [7] present some preliminary results in this direction. Here again, the unique features of HMS machines simplify the computation graphs that are usually created in model checking. In particular, the possibility of a system being in multiple states reduces the size of the computation graph significantly. Also, by using the temporal logic TIL, one can record much more complicated facts at the nodes of a computation graph, further reducing its size. Finally, unlike specification languages based on finite-state machines, no separate language is needed to express the properties to be evaluated. For example, the complications of branching time temporal logics that are often used in model checking can be avoided completely.

SUMMARY

The application of formal methods for specification and verification can have a major impact in improving the state-of-the-art in the design of sequential control systems that control medical instrumentation, transportation systems, nuclear reactors, manufacturing systems, chemical plants, and power distribution systems. Better understanding of requirements and the elimination of design errors are two important benefits of formal methods. The "hierarchical multi-state (HMS) machine" methodology offers a promising approach for the formal specification of sequential control systems. Its major advantages include compactness of expression, a rich language for expressing temporal constraints, a graphic formalism, and a variety of verification techniques. A prototype implementation of HMS machines is currently under development that provides capabilities for hierarchical specification and support for formal verification.

REFERENCES

[1] Chandy, K.M., and C. Kesselman, "Parallel programming in 2001," *IEEE Software,* November 1991, pp. 11-20.

[2] David, R., and H. Alla, *Petri Nets & Grafcet: Tools for Modeling Discrete Event Systems,* Prentice Hall, New York, 1992.

[3] Filer, R., and G. Leinonen, *Programmable Controllers and Designing Sequential Logic,* Saunders College Publishing, Ft. Worth, 1992.

[4] Gabrielian, A., "HMS machines: a unified framework for specification, verification and reasoning for real-time systems," in *Foundations of Real-Time Computing: Formal Specifications and Methods,* A.M. van Tilborg and G.M. Koob (Eds.), Kluwer Academic Publishers, Boston, 1991, pp. 139-166.

[5] Gabrielian, A., and M.K. Franklin, "State-based specification of complex real-time systems," *Proceedings of the 9th Real-Time Systems Symposium,* Huntsville, AL, December 6-8, 1988, pp. 2-11.

[6] Gabrielian, A., and M.K. Franklin, "Multi-level specification of real-time systems," *Communications of the ACM,* Vol. 34, No. 5, May 1991, pp. 50-60.

[7] Gabrielian, A., and R. Iyer, "Integrating automata and temporal logic: A framework for specification and verfication of real-time systems and software," in *The Unified Computation Laboratory: Modelling, Specification, and Tools,* C. Rattray amd R.G. Clark (Eds.), Clarendon Press, Oxford, 1992, pp. 261-273.

[8] Gabrielian, A., and R. Iyer, "Verifying properties of HMS machine specifications of real-time systems," *Computer-Aided Verification*, Lecture Notes in Computer Science No. 575, K.G. Larsen and A. Skou (Eds.), Spring-Verlag, Berlin, 1992, pp. 421-431.

[9] Laprie, J.-C., and B. Littlewood, "Probabilistic assessment of safety-critical software: why and how?," *Communications of the ACM*, Vol. 35, No. 2, Feb. 1992, pp. 13-21.

[10] Liptak, B.G., and K. Venczel, *Instrument Engineers' Handbook*, Chilton Book Company, Radnor, PA, 1985.