Implicit Model Checking of Logic-Based Control Systems

Taeshin Park and Paul I. Barton
Dept. of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139

Increasing automation in the CPI has led to the growing use of logic-based control systems (or safety interlock systems) in safety-related and sequencing operations. The growing complexity and safety-critical nature of typical applications make developing technologies for the formal verification of logic-based control systems with respect to their functionality a crucial and urgent issue. It still remains elusive to analysis, primarily due to the exponential growth of the alternatives that must be examined with application size. Implicit model checking is a formal verification technology that can be applied to the verification of large-scale logic-based control system. Its primary advantage is to formally verify large-scale coupled systems, where a novel and compact model formulation makes tractable previously inaccessible problems. Logic-based control systems are represented compactly as an implicit Boolean state-space model, and the properties to be verified are represented in the language of temporal logic. Verification is posed as a Boolean satisfiability problem and then transformed into its equivalent integer programming feasibility problem, which allows for efficient solution with standard branch-and-bound algorithms. Benefits of the methodology are demonstrated by applying to large-scale industrial examples.

Introduction

Safety and environmental regulations, economic objectives, and manufacturing constraints are encouraging the installation of increasingly sophisticated discrete control systems in the chemical industries. As a class of discrete control systems, logic-based control systems (LCSs) play an important role in safety-related tasks (e.g., safety interlocks) and sequencing tasks (e.g., start-up, shutdown, batch operation). Typical applications include boilers and furnaces, nuclear and chemical reactors, power generation systems, oil and gas platforms, large rotating machinery, synthetic pharmaceutical processes, and fermentation processes.

As an example of a LCS and its underlying process, consider the tank-filling process in Figure 1, and its LCS represented by a *binary logic diagram* (ANSI/ISA, 1981) in Figure 2. The LCS shown is responsible for decisions concerning the running of the pump (the control element). Input signals to the LCS include hand-operated control signals (e.g., HS1, HS2, HS7), and signals that indicate threshold crossings in the system state (e.g., LSH3, LSH4, PSL5). The logic of the control system embeds *shutdown logic* (e.g., the pump is shut down regardless of the system state if the suction pressure remains low for more than 5 s), *permissive logic* (e.g., the pump cannot be started if switches HS1 and HS2 are ON simultaneously), and *sequences* (e.g., open the inlet valve and close the inlet valve on the other tank, wait until the valves have responded, start the pump). Note that even in this simple example, multiple functionalities are embedded in a coupled manner. Due to the coupled embedding of multiple functionalities, modern LCSs exhibit considerable complexity (e.g., a burner management system, the safety system for a toxic and/or exothermic chemical reactor, or the interlock system for the top side of an offshore production platform).

The two key issues in the design of a LCS are *safety integrity* and *functionality* (AIChE/CCPS, 1993). Safety integrity is usually quantified by *availability* and *reliability*, and the desired level of integrity is obtained by the selection of an acceptable *configuration* (e.g., redundancy and voting logic) and *implementation* (e.g., PLC and relays). Much progress on reliable procedures for achieving the desired integrity has been made in recent years, and these methods are now well

Correspondence concerning this article should be addressed to P. I. Barton.

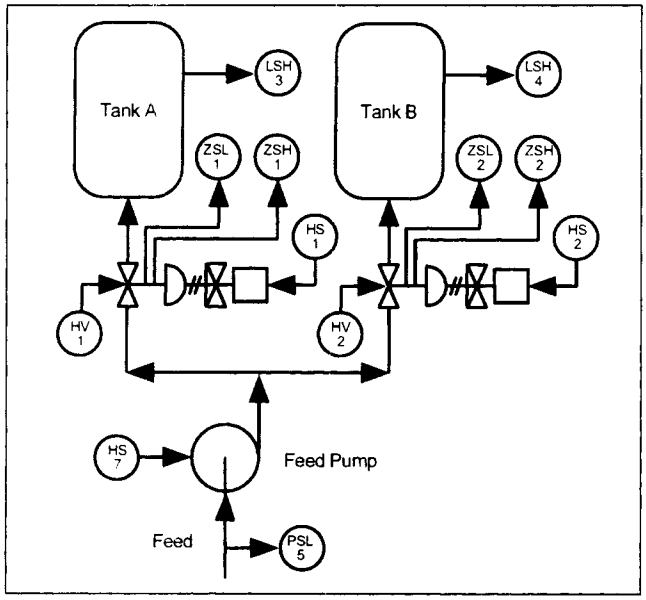

Figure 1. Tank filling process.

established in industry (AIChE/CCPS, 1993). On the other hand, much less attention has been paid to formal design of logic-based control systems with respect to functionality. Figure 3 shows recent data on the primary causes of control system failures in safety-related applications based on 34 incidents (Bell, 1994). According to this survey, the majority of failures are caused by inadequate specification and design of systems. The occurrence of these incidents would be greatly reduced if the original design or design modification could be verified formally against its specifications. For example, the LCS for the tank-filling operation in Figure 2 should be verified against a set of specifications, such as the pump must be shut down if the tank selected for filling becomes full regardless of other system inputs. Our studies and discussions with industry leaders indicate that there is currently an urgent need for the development of a formal and efficient approach to

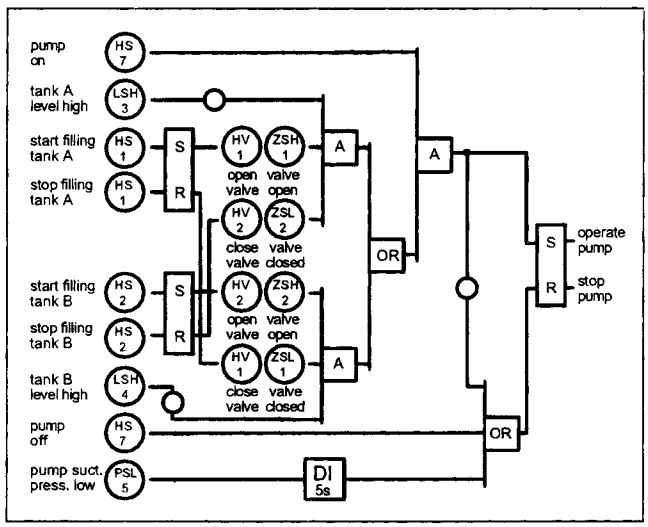

Figure 2. Logic-based control system for tank-filling process.

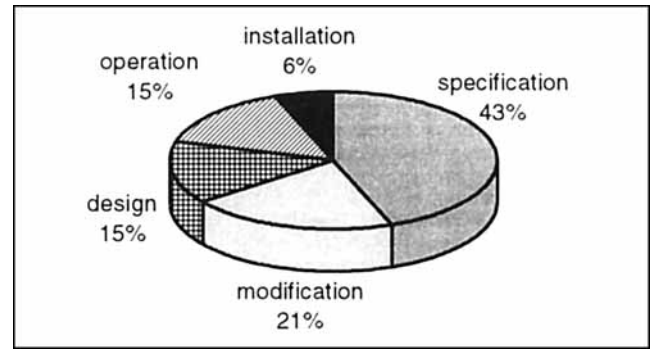

Figure 3. Primary causes of control system failure.

the design and verification of LCSs with respect to functionality.

Thus, the growing complexity of LCS applications and the need for *correct* designs with respect to specifications together with the ad hoc and/or inefficient nature of current industrial practices motivate the study of rigorous approaches to the design and verification of such systems.

Functional design of logic-based control systems

Figure 4 shows the prototypical formal functional design procedure for LCSs that we propose (Park and Barton, 1996). Ideally, the logic design stems from a formal specification of the intended functionality, although at present designs tend to be based on informal information collected from several sources, such as national standards, more detailed company interpretations of these standards, and process-specific documentation. Once the logic has been designed, formal verification against these specifications is highly desirable. A guarantee for complete correctness at least for safety properties is essential. In both cases, the role of a formal implementation-independent representation is pivotal to the rigor of these steps. After successful verification, the system can then be

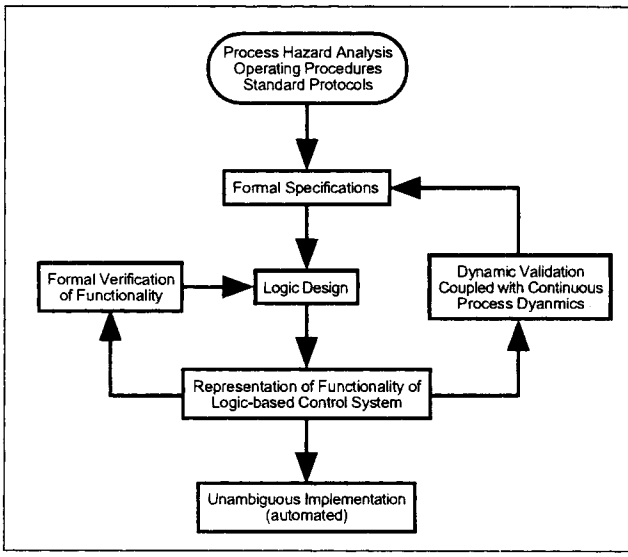

Figure 4. Prototypical functional design procedure for logic-based control systems.

implemented using particular hardware or software (in principle, automatically). Otherwise, the design should be amended to eliminate errors found in the verification step.

As a further step, it is desirable to verify *formally* the functionality of the LCS, considering its dynamic coupling with the underlying chemical process. The notion of *hybrid discrete/continuous dynamic systems* (Barton and Park, 1997) can be used to formulate this problem. However, only a very limited class of hybrid systems can be verified in a formal manner (Alur et al., 1993; Kestne et al., 1993). Therefore, the hybrid system of the LCS and the underlying nonlinear process cannot be verified formally. However, the performance of the LCS with its underlying process can be validated. Hybrid discrete/continuous dynamic simulation technologies (Barton and Park, 1997) can be used to study the overall system response in a set of key upset scenarios identified by a process hazard analysis. The results of such a validation step may identify problems with the original functional specifications, and thus require revision and redesign of the logic.

Formal verification of specifications against a specific logic design is the main topic of this article. Even though there can be different levels of verification or validation, we will focus on formal verification of the implementation-independent functionality of a LCS.

Formal verification of logic-based control systems

As a class of discrete dynamic systems, LCSs exhibit transient behavior between discrete states by processing a sequence of logic signals to produce a series of logical outputs interfaced to control elements in the process. Thus, LCSs can be described as *sequential logic systems*, logic systems whose outputs depend upon both current inputs and the past history of inputs encapsulated in state variables (Friedman, 1986), as shown in Figure 5. Figure 6 depicts state spaces for discrete dynamic systems and continuous dynamic systems. Compared to the infinite state space of continuous dynamic systems, the state space of discrete dynamic systems is finite but combinatorial, that is, the maximum number of discrete states is exponential in the number of state variables, although all may not be reachable in a particular system. This exponential growth of the state space is commonly known as the *state-explosion problem*. Formal verification of LCSs requires all reachable states to be explored (either explicitly or implicitly). Hence, state explosion creates severe problems for formal verification of large-scale systems.

Extensive simulation is the most widely used validation technique in industry. Even though simulation-based methods are relatively straightforward to implement, they are very costly and limited in the extent to which the complete state space can be explored. In particular, formal verification requires that a combinatorial number of states and transitions

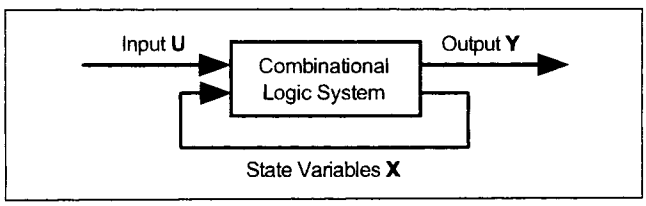

Figure 5. Sequential logic systems.

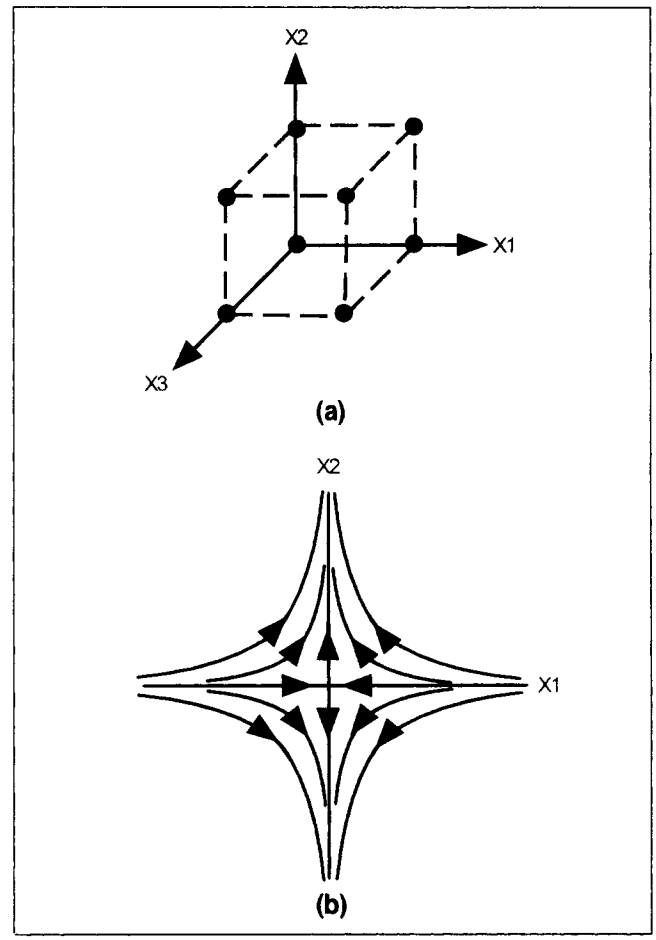

Figure 6. State spaces of discrete and continuous dynamic systems: (a) discrete state space; (b) continuous state space.

must be tested by simulation. For example, exhaustive simulation of a sequential logic system that contains 100 inputs and state variables together will take about 4×10^{12} years even if we assume that each simulation takes 10^{-10} s. Due to the combinatorial number of possible states and transitions, simulation cannot be considered as a viable formal verification technology for the large-scale systems currently being implemented in industry.

Until now, the most successful verification methods for sequential logic systems have been based on *model checking* (Clarke et al., 1986). Model checking is the problem of determining whether a model of a design satisfies a given abstract specification or not. Within this framework, models are described as state transition systems (e.g., state transition graphs) and specifications are expressed in a propositional temporal logic. Most model-checking algorithms are also based on enumeration of the complete state space for the system. Enumeration-based model checking is highly automatic because it can find the set of states where a given specification is true without any user intervention, and the complexity of the algorithm is linear in the size of the state transition graph and in the length of the specification formula. However, model checking suffers from an explosion in the size of the state transition graph for large-scale prob-

lems, since the number of nodes in this graph (states) is exponential in the number of state variables, and the number of edges (transitions) is exponential in the number of inputs and state variables. A more recent development that attempts to mitigate this problem is *symbolic model checking* (Bose and Fisher, 1989; Burch et al., 1990; Coudert et al., 1990). The basic idea is to manipulate sets of states and sets of transitions instead of individual states and individual transitions. The Boolean formulas that symbolize sets of states and sets of transitions are represented by ordered binary decision diagrams (OBDDs) (Bryant, 1986), which are often much more compact than explicit representations because they capture some of the regularity in the state space. While the idea of symbolic model checking and various refinements of the OBDD-based techniques have greatly increased the size of the problems that can be verified to over 400 state variables and 10^{120} states (Burch et al., 1991, 1994), many practical problems are still too large to be verified. It is therefore important to find techniques to extend the size of the problems that can be verified (Clarke et al., 1993). Both model checking and symbolic model checking have been applied with success to several small- to medium-scale verification problems in the chemical industries (Moon et al., 1992; Moon, 1994; Probst and Powers, 1994; Probst et al., 1995, 1997; Jeon et al., 1995), although these problems probably represent the size limit of current verification technology.

In this article, we present a novel model-checking technique that in particular can be applied to systems of realistic sizes encountered in the chemical processing industries. Any formal verification technique raises the problems of model representation, specification language, and verification method. In our approach, the model for sequential logic systems is represented implicitly in terms of a system of Boolean equations, and specifications are expressed in a subset of propositional branching-time temporal logic. Due to the implicit nature of the model representation, model formulation is not combinatorial, which increases the problem size that can be formulated dramatically. An efficient verification method based on the implicit enumeration implemented in standard integer programming techniques is employed to determine automatically if the specifications are satisfied by the model. As a result, our method does not involve explicit enumeration of the full state space. The article is organized as follows. The next section describes the implicit model for sequential logic systems, which is followed by the description of the class of temporal logic currently supported by our approach for specification. Our verification technique is given in the fourth section, followed by an industrial-scale example.

The Model

In this section, we present an implicit Boolean state-space model for LCSs.

Modeling time

With reference to Figure 2, LCSs are passive in the sense that they do not manipulate control elements in the process unless an external *event* occurs, which can be a transition in explicit control signals (e.g., "Start Filling Tank A" is pressed) or a predefined threshold crossing by the continuous state of the process (e.g., "Tank A Level High" becomes TRUE). The system will then activate, calculate a transition in the output signals to the control elements (e.g., "Operate Pump" becomes FALSE), and then become passive until activated again in a similar manner. Subject to this observation, the notion of physical time is replaced by the simpler notion of order among external events.

Furthermore, comparing the relative input frequencies and the input–output response times of a LCS to those of a chemical process, it is valid to assume that any LCS will finish its dynamic response to any external event before any further external event occurs. More formally, we assume that the LCS operates in a *fundamental mode*, which means that input signals are never changed unless the system is in a stable state (Friedman, 1986). The system has reached a *stable* (or steady) state if none of the signals are changing given a fixed set of values for the input signals. Based on this assumption, we only consider stable states.

Based on these observations, the physical continuous time domain is discretized at the points of external events or, equivalently, transitions in the input signals. Therefore, time intervals between events will not necessarily be uniform. Note that this timing convention will result in the minimum number of discretization points while capturing all the relevant details, and timing is determined by the operation being performed by the overall system rather than an irrelevant uniform clock signal. Finally, the timing diagram of Figure 7 represents the convention for mapping between discrete and continuous time domains. For example, if the variable X experiences a transition from FALSE to TRUE at time t_k, the value of X at t_k is taken to be TRUE rather than FALSE.

Implicit Boolean state-space model

The key property that distinguishes our modeling framework from previous efforts is the novel notion of *implicitness*, in the sense that the model encapsulates only the relevant set of time-invariant relationships between state variables, rather than a partial or full enumeration of the state space. Hence, our model is analogous to the state-space (or ODE) model of a continuous dynamic system. This implicitness leads to a compact representation of the system dynamics and avoids the problem of state explosion during model formulation.

As mentioned earlier, LCSs can be described as sequential logic systems whose current states depend upon both current inputs and the previous state. The sequential logic system to be verified is modeled as a *deterministic* finite-state machine (FSM). A system is said to be deterministic if a given sequence of inputs always produces the same sequence of out-

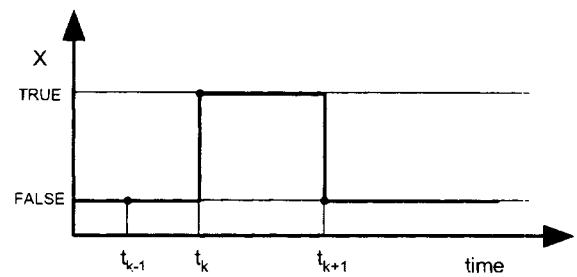

Figure 7. Mapping between discrete and continuous time domains.

puts. The behavior of the FSM is described by a vector of n Boolean state variables $X \in \{F,T\}^n$, a vector of m Boolean inputs $U \in \{F,T\}^m$, and a vector of l Boolean outputs $Y \in \{F,T\}^l$. Each discrete point in state space is encoded by an assignment of Boolean values to the vector of state variables. The *state transition equation* (Eq. 1a) and *output equation* (Eq. 1b) for the FSM are given as a system of Boolean equations:

$$X_k \leftrightarrow f(X_{k-1}, U_k) \tag{1a}$$

$$Y_k \leftrightarrow g(X_k, U_k) \tag{1b}$$

where X_k and X_{k-1} encode the current state and the previous state, respectively; U_k and Y_k encode the current input and output, respectively; and f:{F, T}$^n \times$ {F, T}$^m \rightarrow$ {F, T}^{n} and g:{F, T}$^n \times$ {F, T}$^m \rightarrow$ {F, T}^{l} are vectors of logical propositions. The state transition equation characterizes a transition from one state to another for a particular valuation of the inputs, and the output equation calculates outputs for the transition. Note that the FSM described by Eq. 1 is completely specified. In other words, the current state X_k and output Y_k are uniquely determined for any pair of (X_{k-1}, U_k). The analogy between Eq. 1 and the familiar differential or difference equation models is evident.

Definition 1. The transition from X_{k-1} to X_k is *legal* if the transition satisfies the state transition equation of Eq. 1a.

Definition 2. The state X is a *valid* state if it is reachable through legal transitions.

Definition 3. The state X is an *initial* state if it is unreachable from any other states but there exist legal transitions from this state.

The initial states represent states that the system enters at the beginning such as power-up states. Note that initial states must be specified explicitly in order to include them in the verification because they cannot be reached by any transition.

By definition, any feasible solution that satisfies Eq. 1 represents a legal transition between valid states or from an initial state. Therefore, the Boolean state-space model of Eq. 1 contains *implicitly* all possible legal transitions between valid states or from initial states. Note that the model is compact because it includes only the relevant set of invariant relationships between variables. However, an additional algorithmic procedure is necessary to draw out explicit information such as a particular sequence of transitions.

As an example, consider the tank interlock system (Victor, 1979) represented as a binary logic diagram in Figure 8. The tank-filling operation is aborted by closing on–off valve SV430 if there is a high-pressure alarm (PAH430) or the Stop button is pressed, and can be resumed by pressing the Reset button. The signal SV430 is identified as a state variable due to the existence of the feedback path, and the signals PAH430, Stop, and Reset are inputs. The state transition equation is obtained directly from the logic in Figure 8 as

$$X_{1,k} \leftrightarrow (\neg U_{1,k} \wedge \neg U_{2,k}) \wedge (U_{3,k} \vee X_{1,k-1}), \tag{2}$$

where X = [SV430] and U = [PAH430, Stop, Reset]. The symbols {$\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$} are standard Boolean operators (or connectives) meaning {NOT, AND, OR, IMPLICATION (IF ... THEN), EQUIVALENCE}, respectively. Note that there is no output equation in this example. Figure 9 is a

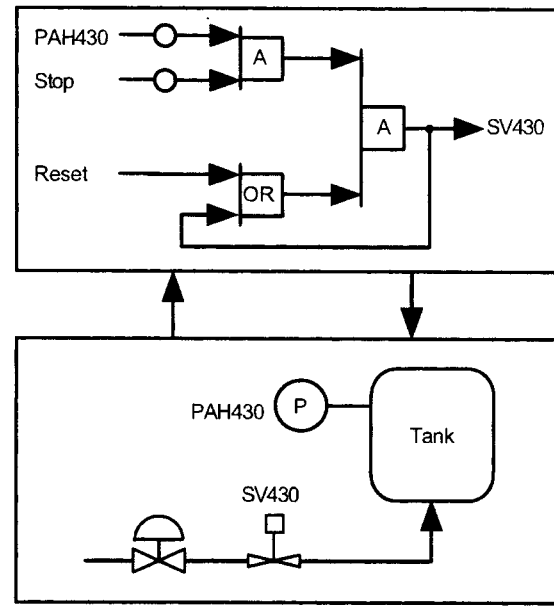

Figure 8. Tank interlock system with its underlying process.

state transition graph for this system. A node represents a particular valuation of state variables, and the edge represents a transition from one state to another for a particular valuation of the inputs. A feasible solution of Eq. 2 corresponds to an edge in the state transition graph. For example, the edge F $\xrightarrow{\text{FFT}}$ T represents the feasible solution X_{k-1} = [F], X_k = [T], U_k = [FFT] of Eq. 2. Substituting this feasible solution into Eq. 2, it is not difficult to see that the current state is uniquely determined by the current input alone. On the other hand, this is not the case for the transition T $\xrightarrow{\text{FFF}}$ T or F $\xrightarrow{\text{FFF}}$ F. In this case, the current input cannot determine the current state uniquely. Substituting U_k = [FFF] into Eq. 2 yields $X_{1,k} \leftrightarrow X_{1,k-1}$. Therefore, the current state keeps the value of the previous state.

Subject to this observation, the transitions can be categorized into *retentive* and *nonretentive* transitions:

Definition 4. The transition $X_{k-1} \xrightarrow{U_k} X_k$ is *nonretentive* if X_k is uniquely determined by U_k regardless of X_{k-1}. Otherwise, it is *retentive*. If U_k represents a retentive transition, at least one element of the state transition equation (Eq. 1a) becomes $X_{i,k} \leftrightarrow X_{i,k-1}$, where $i \in \{1, \ldots, n\}$.

The term *retentive* is adopted because these transitions retain partial or complete memory of the previous state of the system.

The objective of our modeling is to construct a model that contains implicitly all possible legal transitions between valid

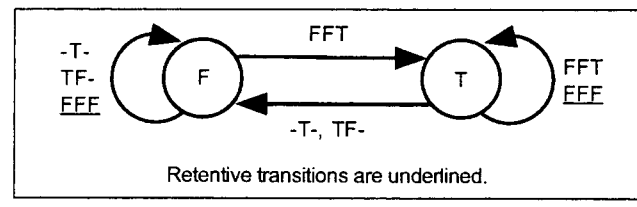

Figure 9. State transition graph for tank interlock system.

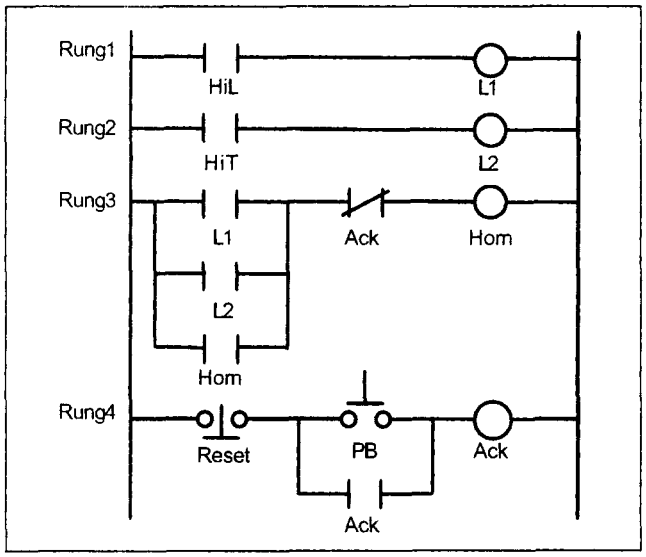

Figure 10. Alarm acknowledge system.

states or from initial states. The Boolean state-space model of Eq. 1 satisfies this objective. However, as discussed in the preceding section, we are only interested in stable states. The original model of Eq. 1 does not satisfy this requirement because it includes both stable and unstable states, which are defined as follows:

Definition 5. The state $\tilde{X}$ is a *stable* state if it is the fixed point of Eq. 1a for given unchanging inputs U, that is, $\tilde{X} \leftrightarrow f(\tilde{X}, U)$. Otherwise, it is an *unstable* state. The stable states correspond to the nodes with self-edges in the state transition graphs.

As an example, consider an alarm-acknowledge system (Moon et al., 1992) represented as a *ladder logic diagram* (Otter, 1988) in Figure 10. The signals Ack and Horn are identified as state variables because of the latching of the signals in rungs 3 and 4, respectively, and the signals HiL, HiT, Reset, and PB are inputs. The Boolean state-space model can again be obtained directly from Figure 10 as

$$\begin{aligned} X_{1,k} &\leftrightarrow \neg U_{3,k} \wedge (U_{4,k} \vee X_{1,k-1}) \\ X_{2,k} &\leftrightarrow \neg X_{1,k-1} \wedge (U_{1,k} \vee U_{2,k} \vee X_{2,k-1}), \end{aligned} \tag{3}$$

where $X = [\text{Ack, Horn}]$ and $U = [\text{HiL, HiT, Reset, PB}]$. Figure 11 is a state transition graph representing all the feasible solutions of Eq. 3. The state $X = [\text{TT}]$ is unstable because there is no valuation of the inputs that makes this state a fixed point of Eq. 3. It is therefore necessary to revise the original Boolean state-space model to exclude any unstable states.

The revised implicit Boolean state-space model is

$$\tilde{X}_k \leftrightarrow f(\tilde{X}_k, U_k) \tag{4a}$$

$$\tilde{X}_{k-1} \leftrightarrow f(\tilde{X}_{k-1}, U_{k-1}) \tag{4b}$$

$$h(\tilde{X}_k, U_k) \rightarrow \left(\tilde{X}_k \leftrightarrow \tilde{X}_{k-1}\right) \tag{4c}$$

$$Y_k \leftrightarrow g(\tilde{X}_k, U_k), \tag{4d}$$

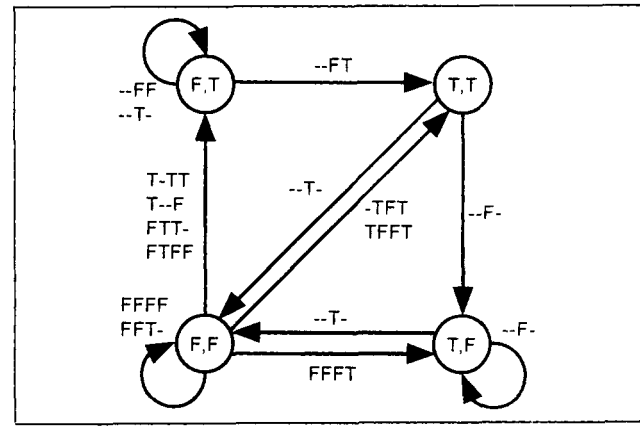

Figure 11. State transition graph for alarm acknowledge system defined by Eq. 3.

where Eqs. 4a–4c are state transition equations, and the output equations, Eq. 4d, are unchanged. Equation 4a (or Eq. 4b) restricts the valuations of $\tilde{X}_k$ (or $\tilde{X}_{k-1}$) to stable states by definition. Note that the value of U_{k-1} is not limited to any particular valuation since we want $\tilde{X}_{k-1}$ to represent all possible stable states. Note that nonretentive transitions are uniquely defined by Eqs. 4a and 4b. If U_k represents a retentive transition, then at least one element of Eq. 4a will be $\tilde{X}_{i,k} \leftrightarrow \tilde{X}_{i,k}$, where $i \in \{1, \ldots, n\}$, which can be satisfied by any valuation of $\tilde{X}_{i,k}$. Therefore, $\tilde{X}_k$ is not uniquely determined by Eqs. 4a and 4b in this case. An additional constraint is necessary to determine $\tilde{X}_k$ uniquely for retentive transitions. Equation 4c determines all retentive transitions uniquely. The Boolean function h:$\{\text{F,T}\}^n \times \{\text{F,T}\}^m \rightarrow \{\text{F,T}\}^n$ is a vector of logical propositions called *retentive functions* that define conditions for retentive transitions. The retentive functions h can be derived from f automatically, which is discussed later.

As an example, consider again the alarm-acknowledge system. The revised implicit Boolean state-space model can be derived from Eq. 3 as follows:

$$\tilde{X}_{1,k} \leftrightarrow \neg U_{3,k} \wedge (U_{4,k} \vee \tilde{X}_{1,k}) \tag{5a}$$

$$\tilde{X}_{2,k} \leftrightarrow \neg \tilde{X}_{1,k} \wedge (U_{1,k} \vee U_{2,k} \vee \tilde{X}_{2,k}) \tag{5b}$$

$$\tilde{X}_{1,k-1} \leftrightarrow \neg U_{3,k-1} \wedge (U_{4,k-1} \vee \tilde{X}_{1,k-1}) \tag{5c}$$

$$\tilde{X}_{2,k-1} \leftrightarrow \neg \tilde{X}_{1,k-1} \wedge (U_{1,k-1} \vee U_{2,k-1} \vee \tilde{X}_{2,k-1}) \tag{5d}$$

$$(\neg U_{3,k} \wedge \neg U_{4,k}) \rightarrow \left(\tilde{X}_{1,k} \leftrightarrow \tilde{X}_{1,k-1}\right) \tag{5e}$$

$$\neg \tilde{X}_{1,k} \wedge \neg (U_{1,k} \vee U_{2,k}) \rightarrow \left(\tilde{X}_{2,k} \leftrightarrow \tilde{X}_{2,k-1}\right). \tag{5f}$$

Equations 5a–5d define all nonretentive transitions, and Eqs. 5e and 5f define all retentive transitions. Figure 12 is a state transition graph representing all feasible solutions of Eqs. 5a–5d. Note that the unstable state $X = [\text{TT}]$ is excluded. However, retentive transitions are not uniquely determined. For example, the transition from $\tilde{X}_{k-1} = [\text{TF}]$ when $U_k = [\text{FFTT}]$ is not unique. Instead, the transition can be $\text{TF} \xrightarrow{\text{FFTT}} \text{FF}$ or $\text{TF} \xrightarrow{\text{FFTT}} \text{FT}$. However, we know that the correct transition is $\text{TF} \xrightarrow{\text{FFTT}} \text{FF}$ because $X_k = [\text{F}, X_{2,k-1}]$ by

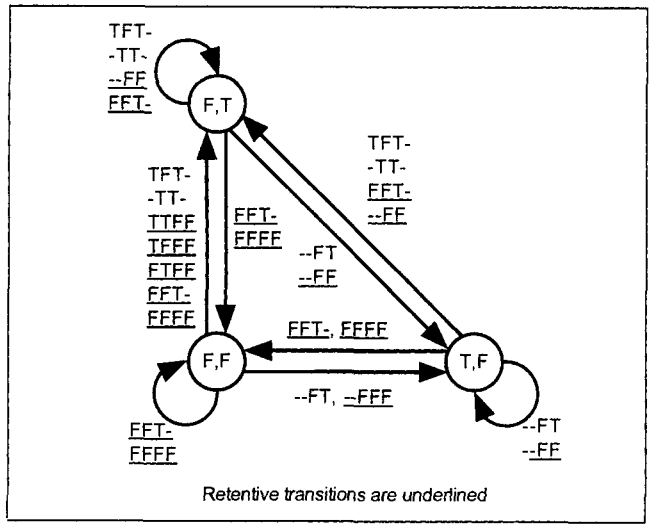

Figure 12. State transition graph for alarm acknowledge system defined by Eqs. 5a–5d.

substituting $U_k = [\text{FFTT}]$ into Eq. 3. By adding Eqs. 5e and 5f, all retentive transitions are determined uniquely. Figure 13 shows a state transition graph defined by the complete state-space model of Eq. 5.

The Boolean state-space model of Eq. 4 now embeds implicitly all possible legal transitions between stable states. However, it does not embed transitions from initial states if these initial states are not stable. Figure 14 illustrates this situation where the state $X = [\text{FF}]$ corresponds to an unstable initial state. The model of Eq. 4 does not include the transition $\text{FF} \xrightarrow{\text{FT}} \text{TF}$ because $\tilde{X}_{k-1}$ in Eq. 4 does not include this unstable state. However, this is not a limitation because the initial states must be specified explicitly as mentioned earlier.

Coupled with an explicit specification of the initial states, the Boolean state-space model of Eq. 4 or $v(U_{k-1}, U_k, \tilde{X}_{k-1}, \tilde{X}_k, Y_k)$ embeds implicitly all possible legal transitions between stable states or from initial states. Hence, any feasible

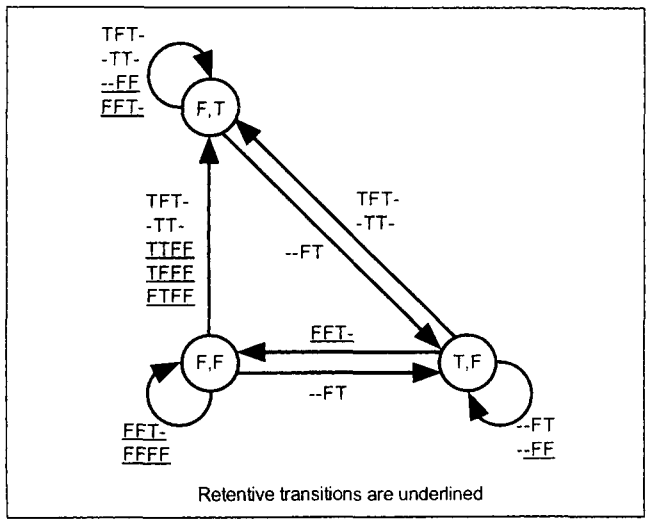

Figure 13. State transition graph for alarm acknowledge system defined by Eq. 5.

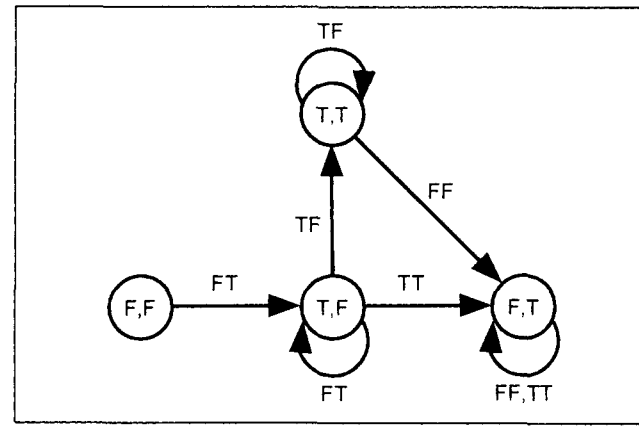

Figure 14. FSM containing unstable initial states.

solution of the model represents a particular legal transition between stable states or from initial states. This model forms the basis for our large-scale verification technology. Note that the size of the model in terms of the number of logical propositions is linear in the number of state variables and the number of outputs.

Finally, from the examples it is clear that the system of Boolean equations (Eq. 1) can be obtained automatically from standard industrial representations for LCSs such as binary logic diagrams, ladder logic diagrams, and programming languages. Even though the model of Eq. 1 or Eq. 4 is deterministic by assumption, the mapping from standard representations to our Boolean state-space model is not necessarily unique. For example, binary logic diagrams that have nonunique feedback cutsets yield a family of Boolean state-space models, each of which may have a different functionality due to the existence of *hazards* and *races* (Park and Barton, 1996). It is therefore necessary to guarantee that any logic design has had any potential ambiguities resolved before applying implicit model checking [e.g., by explicit specification of the feedback cutset for a binary logic diagram (Park and Barton, 1997)]. Clearly, any potential for ambiguity is highly undesirable in safety-related applications and should be resolved by any design procedure (Park and Barton, 1996).

Derivation of retentive functions

The retentive functions in Eq. 4c can be derived by examining the conditions under which the inputs do not determine the state vector uniquely. The transition equation for ith state variable $\tilde{X}_{i,k}$ is

$$\tilde{X}_{i,k} \leftrightarrow f_i(\tilde{X}_k, U_k). \tag{6}$$

First, the transition function f_i can be transformed into the form:

$$\tilde{X}_{i,k} \leftrightarrow f_{i,c1}(\tilde{X}'_k, U_k) \wedge [f_{i,c2}(X'_k, U_k) \vee \tilde{X}_{i,k}], \tag{7}$$

where $\tilde{X}'_k$ is a vector of state variables excluding $\tilde{X}_{i,k}$, and the Boolean formulas $f_{i,c1}$, $f_{i,c2}$ are in *conjunctive normal form* (a conjunction of clauses where each *clause* is a disjunction of atomic propositions). A *conjunction* is a set of logical

propositions connected by the AND operator $\wedge$, while a *disjunction* is a set of logical propositions connected by the OR operator $\vee$. An *atomic* proposition is a logical proposition that does not contain any Boolean connectives. An algorithm to convert Boolean formulas into conjunctive normal forms (e.g., Clocksin and Mellish, 1984) can be used with a distributive law to derive Eq. 7 automatically from Eq. 6. For example, consider the transition equation for $\tilde{X}_1$:

$$\tilde{X}_{1,k} \leftrightarrow (U_{1,k} \wedge U_{2,k}) \vee (\neg U_{3,k} \wedge \tilde{X}_{1,k}). \quad (8)$$

First, the transition function for $\tilde{X}_1$ can be converted into a conjunctive normal form:

$$\tilde{X}_{1,k} \leftrightarrow (U_{1,k} \vee \neg U_{3,k}) \wedge (U_{2,k} \vee \neg U_{3,k}) \wedge (U_{1,k} \vee \tilde{X}_{1,k}) \wedge (U_{2,k} \vee \tilde{X}_{1,k}). \quad (9)$$

Factorizing the term $\tilde{X}_{1,k}$ on the righthand side of Eq. 9, the expression corresponding to Eq. 7 is obtained as

$$\tilde{X}_{1,k} \leftrightarrow (U_{1,k} \vee \neg U_{3,k}) \wedge (U_{2,k} \vee \neg U_{3,k}) \wedge ((U_{1,k} \wedge U_{2,k}) \vee \tilde{X}_{1,k}). \quad (10)$$

It is clear from Eq. 7 that $\tilde{X}_{i,k}$ cannot be determined uniquely if $f_{i,c1}(U_k, \tilde{X}'_k)$ and $\neg f_{i,c2}(U_k, \tilde{X}'_k)$. Therefore, the memory-retaining constraint for ith state variable $\tilde{X}_{i,k}$ is

$$[f_{i,c1}(U_k, \tilde{X}'_k) \wedge \neg f_{i,c2}(U_k, \tilde{X}'_k)] \rightarrow (\tilde{X}_{i,k} \leftrightarrow \tilde{X}_{i,k-1}). \quad (11)$$

Note that derivation of memory-retaining propositions has the same computational complexity as converting Boolean formulas into conjunctive normal form. In particular, the number of clauses in the conjunctive normal form can be exponential in the worst case. However, in general the number of clauses in the conjunctive normal form can be made linear in the number of Boolean variables in the original proposition by introducing a polynomial number of intermediate variables, using an algorithm such as *structure-preserving clause form translation* (Plaisted and Greenbaum, 1986; Boy de la Tour, 1990; Hähnle, 1993).

As an example, consider again the tank interlock system of Figure 8. The righthand side of Eq. 2 is already in the form of Eq. 7 (replace $X_{1,k-1}$ and $X_{1,k}$ with $\tilde{X}_{1,k}$ in Eq. 2). Therefore, the retentive function is

$$(\neg U_{1,k} \wedge \neg U_{2,k}) \wedge \neg U_{3,k} \rightarrow (\tilde{X}_{1,k} \leftrightarrow \tilde{X}_{1,k-1}), \quad (12)$$

and the complete Boolean state-space model for the tank interlock system of Figure 8 is obtained as

$$\tilde{X}_{1,k} \leftrightarrow (\neg U_{1,k} \wedge \neg U_{2,k}) \wedge (U_{3,k} \vee \tilde{X}_{1,k})$$
$$\tilde{X}_{1,k-1} \leftrightarrow (\neg U_{1,k-1} \wedge \neg U_{2,k-1}) \wedge (U_{3,k-1} \vee \tilde{X}_{1,k-1})$$
$$(\neg U_{1,k} \wedge \neg U_{2,k}) \wedge \neg U_{3,k} \rightarrow (\tilde{X}_{1,k} \leftrightarrow \tilde{X}_{1,k-1}). \quad (13)$$

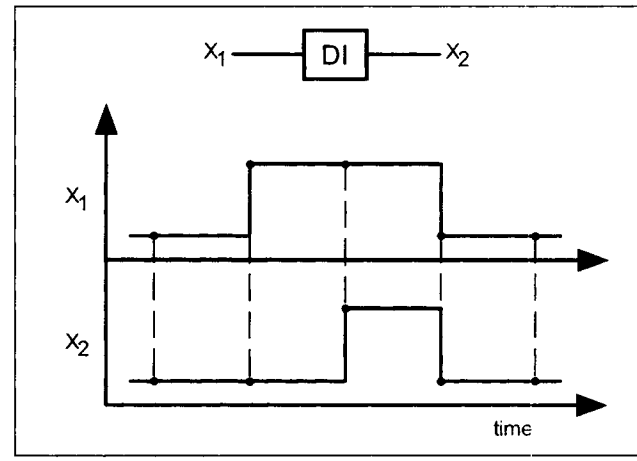

Figure 15. Timing diagram of delay-on timers.

Modeling unit delay timers

All delay timers are abstracted and modeled as unit delay timers. Abstract logical relationships between input and output signals are established by stating the minimum set of relevant properties of delay timers. For example, Figure 15 represents the timing diagram of unit delay-on timers. The abstract logical proposition that represents the functionality of the unit delay-on timer is

$$(\neg X_{1,k} \wedge \neg X_{2,k}) \vee (\neg X_{1,k-1} \wedge X_{1,k} \wedge \neg X_{2,k}) \vee (X_{1,k-1} \wedge X_{1,k} \wedge X_{2,k}). \quad (14)$$

Similarly, Figure 16 represents the timing diagram of unit delay-off timers, and the abstract logical relationship is given by

$$(X_{1,k} \wedge X_{2,k}) \vee (X_{1,k-1} \wedge \neg X_{1,k} \wedge X_{2,k}) \vee (\neg X_{1,k-1} \wedge \neg X_{1,k} \wedge \neg X_{2,k}). \quad (15)$$

From the simulation point of view, this abstraction is obviously a loss of information. However, from the verification point of view, the modeling of delay timers as unit delay timers

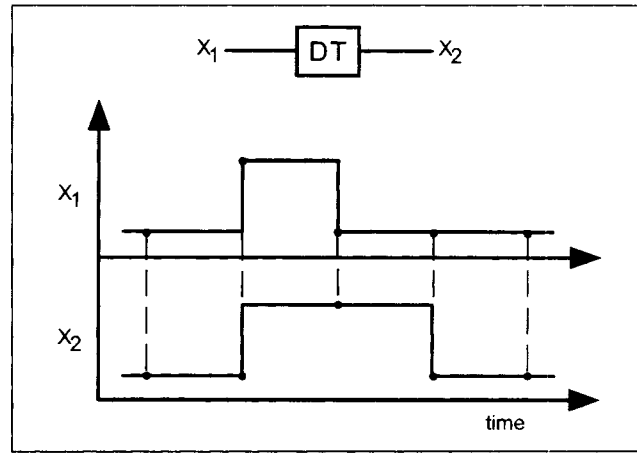

Figure 16. Timing diagram of delay-off timers.

will be adequate because the model-checking algorithm explores the entire state space, including the variables representing the abstract behavior of delay timers. If the specification is satisfied by the model, the model is correct with respect to the specification regardless of the magnitude of delays in timers because we consider all possible combinations of the inputs and outputs for the timers. Thus, the model is verified. If the specification is violated, it is necessary to analyze the problem with more rigorous delay timer models to reach a conclusion concerning correctness of the logic.

Specification Language

The LCS embeds multiple functionalities coupled with each other, and there are large numbers of *correctness properties* that must be satisfied by the LCS model. These properties can be categorized into three classes: shutdown logic, permissive logic, and sequences. For example, consider the tank-filling process of Figure 1. The LCS should always issue a Stop signal to the pump if the level of either tank A or B is higher than its threshold limit (shutdown logic). Similarly, the pump should not operate if an operator tries to fill tank A and B simultaneously (permissive logic).

Figure 17 shows a network of pipes and valves, which provides a sequence of an O_2 stream and then a CH_4 stream to the downstream process. Mixing of O_2 and CH_4 is not allowed to prevent an explosion. Therefore, the LCS for this process must ensure that the O_2 valve and CH_4 valve should not be open at the same time (permissive logic). Furthermore, the LCS must ensure that the N_2 valve should be open just after the O_2 valve or CH_4 valve is closed (sequence).

Finally, consider the typical LCS implemented for a furnace. If unsafe conditions occur or stop signals exist, then the LCS should issue a shutdown signal immediately regardless of the current state (shutdown logic). If the furnace is in a shutdown state, then the furnace should stay in the shutdown state or the furnace can move only into the purge state (sequence). The furnace cannot be ignited unless the furnace has been purged (permissive logic).

In order to verify the LCS model, it is necessary to represent all of these diverse properties in a formal manner, which is a *formal specification*. Note that every individual run or *computation* of the LCS yields a sequence of states and associated transitions, and specifications for the LCS are in general satisfied by some sequences, and not satisfied by some other sequences. *Temporal logic*, a logic of propositions whose truth and falsity may depend on time, is used to represent specifications because the language of temporal logic provides operators for reasoning about computations.

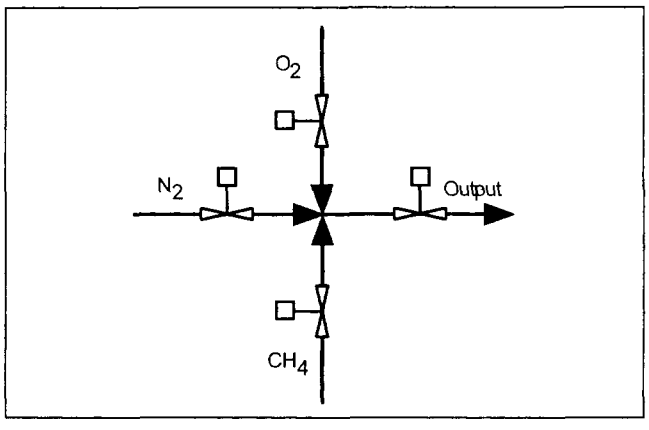

Figure 17. Network of pipes and valves.

Among different classes of temporal logic, the subset of the language of temporal logic of Clarke and Emerson (1981), which is a class of *propositional branching-time temporal logic* (Emerson, 1990), is used for formal specification. The scope of temporal formulas employed in this article will be confined to those involving only two time steps, because our Boolean state-space model is first order. Note that the class of temporal formulas that satisfy these conditions will be sufficient to specify most intended properties of LCSs because the range of properties that can be specified is quite broad (Halbwachs et al., 1989; Jagadeesan et al., 1995).

The temporal formulas are built up from atomic propositions (encoded by inputs, outputs, or state variables), Boolean connectives $\{\neg, \wedge, \vee\}$, and temporal operators $\{\boldsymbol{G}$:always, $\boldsymbol{F}$:sometimes, $\boldsymbol{N}$:next$\}$ restricted by quantifiers $\{\forall, \exists\}$, where $\forall$ (for all) and $\exists$ (there exists) are universal and existential quantifiers, respectively. The syntax of the temporal logic is formally defined as:

- Any atomic proposition is a formula.
- If p, q are formulas, then so are the formulas $\neg p$, $p \wedge q$, and $p \vee q$.
- If p is a formula, then so are the formulas $\forall \boldsymbol{G}(p)$, $\exists \boldsymbol{F}(p)$, $\forall \boldsymbol{N}(p)$, and $\exists \boldsymbol{N}(p)$.

The semantics of the temporal formulas can be formally defined with respect to the model $v(U_{k-1}, U_k, \tilde{X}_{k-1}, \tilde{X}_k, Y_k)$. The formula $\forall \boldsymbol{G}(p)$ (always p) means that the proposition p holds for all states and their associated transitions in the computation. As an example, consider the tank interlock system of Figure 8. One of the requirements for this interlock system is that the inlet valve should be closed (SV430 is FALSE) whenever there is a shutdown signal (PAH430 or Stop) even if the operator presses the Reset button by mistake. Since this property should always be satisfied regardless of the current state of the system, it can be specified formally as $\forall \boldsymbol{G}(U_{1,k} \vee U_{2,k} \rightarrow \neg \tilde{X}_{1,k})$. It is not difficult to see from Figure 9 that any transition from any state satisfies the proposition $U_{1,k} \vee U_{2,k} \rightarrow \neg \tilde{X}_{1,k}$, therefore, the formula $\forall \boldsymbol{G}(U_{1,k} \vee U_{2,k} \rightarrow \neg \tilde{X}_{1,k})$ is TRUE for the model of Figure 9.

The formula $\exists \boldsymbol{F}(p)$ (sometimes p) means that there is some state and its associated transition in the computation at which p holds. Consider the alarm-acknowledge system of Figure 10 as an example. If there is an alarm (Horn is TRUE), then the operator can acknowledge the alarm (Ack is TRUE) and turn it off (Horn is FALSE) by pressing the PB button. Therefore, we can expect that the signals Ack and Horn cannot be TRUE at the same time, which can be formally specified as $\neg \exists \boldsymbol{F}(\tilde{X}_1 \wedge \tilde{X}_2)$. It is easy to see from Figure 13 that the proposition $\tilde{X}_1 \wedge \tilde{X}_2$ is not satisfied, therefore the formula $\exists \boldsymbol{F}(\tilde{X}_1 \wedge \tilde{X}_2)$ [or $\neg \exists \boldsymbol{F}(\tilde{X}_1 \wedge \tilde{X}_2)$] is FALSE (or TRUE) for the model of Figure 13. Note that checking the formula $\neg \exists \boldsymbol{F}(\tilde{X}_1 \wedge \tilde{X}_2)$ is equivalent to checking the formula $\forall \boldsymbol{G}[\neg(\tilde{X}_1 \wedge \tilde{X}_2)]$. In general, the operators $\forall \boldsymbol{G}$ and $\exists \boldsymbol{F}$ are dual: $\forall \boldsymbol{G}(p) = \neg \exists \boldsymbol{F}(\neg p)$.

The formula $\forall \boldsymbol{N}(p)$ (p at all next time) means that the proposition p holds in every immediate successor of the current state, while $\exists \boldsymbol{N}(p)$ (p at some next time) means that the

proposition p holds in some immediate successor of the current state. For example, consider again the alarm-acknowledge system of Figure 10. Note that once there is an alarm (Horn is TRUE), then either it should stay on or it is turned off (Horn is FALSE) only by pressing the PB button. This requirement can be formally specified using $\forall G$ and $\forall N$ as $\forall G\{\tilde{X}_{2,k} \rightarrow \forall N[\tilde{X}_{2,k} \vee (\neg \tilde{X}_{2,k} \wedge U_{4,k})]\}$. It is not difficult to see from Figure 13 that this formula is TRUE by checking the transitions from the state $X = [\mathrm{FT}]$. Note that the operators $\forall N$ and $\exists N$ are dual: $\forall N(p) = \neg \exists N(\neg p)$.

Verification Algorithm

The verification of sequential logic systems against specifications is formulated as a Boolean satisfiability problem, which is then solved as its equivalent integer programming feasibility problem.

Transformation into satisfiability problem

The verification algorithm takes as inputs the model and a specification formula to be verified. The specification formula is transformed into a form that involves only existential quantifiers by removing universal quantifiers using the identities in the preceding section, and then coupled with the model to yield a *Boolean satisfiability problem* (Nemhauser and Wolsey, 1988):

$$\exists\left[U_{k-1}, U_k, \tilde{X}_{k-1}, \tilde{X}_k, Y_k\right]$$

s.t.

$$v\left(U_{k-1}, U_k, \tilde{X}_{k-1}, \tilde{X}_k, Y_k\right)$$

$$w\left(U_k, \tilde{X}_{k-1}, \tilde{X}_k, Y_k\right), \tag{16}$$

which determines if any feasible solutions that satisfy all the constraints exist. The specification constraint $w(U_k, \tilde{X}_{k-1}, \tilde{X}_k, Y_k)$ can be derived from the specification formula by removing all universal quantifiers. Table 1 shows a list of specification formulas and corresponding specification constraints. The only remaining problem is to find out whether the problem in Eq. 16 is satisfiable or not.

Table 1. Specification Formulas and Constraints

Specification Formula	$w(U_k, \tilde{X}_{k-1}, \tilde{X}_k, Y_k)$	Interpretation
$\forall N(p), \tilde{X}_{k-1} = X_0$	$\neg p(\tilde{X}_k), \tilde{X}_{k-1} = X_0$	p holds at every next state of X_0
$\exists N(p), \tilde{X}_{k-1} = X_0$	$p(\tilde{X}_k), \tilde{X}_{k-1} = X_0$	p holds at some next state of X_0
$\forall G(p)$	$\neg p(\tilde{X}_k)$	p holds always
$\forall G[p \rightarrow \forall N(q)]$	$p(\tilde{X}_{k-1}) \wedge \neg q(\tilde{X}_k)$	It always holds that if p occurs, then q holds at every next state
$\exists F(p)$	$p(\tilde{X}_k)$	It is possible that p holds at some states
$\exists F[p \rightarrow \exists N(q)]$	$\neg p(\tilde{X}_{k-1}) \vee q(\tilde{X}_k)$	It is possible that if p occurs, then q holds at some next state

The results of the satisfiability problem should be analyzed differently depending upon the types of quantifiers by which the formula is preceded. For a formula with a universal quantifier, satisfiability means that the formula is FALSE and the valuations that make the problem satisfiable represent *counterexamples*, whereas unsatisfiability means that the formula is TRUE. For a formula with an existential quantifier, satisfiability means that the formula is TRUE and the valuations that make the problem satisfiable represent *witnesses*, whereas unsatisfiability means the formula is FALSE. Note that it is not necessary to have an additional algorithm to construct counterexamples or witnesses since the feasible solutions of the Boolean satisfiability problem are counterexamples or witnesses depending upon the formula verified. Furthermore, the solution algorithm presented in the next subsection can find all feasible solutions systematically.

For example, the formula $\forall G(\neg U_{1,k} \wedge \neg U_{2,k} \rightarrow \tilde{X}_{1,k})$ checks whether the valve will be open if there is no pressure alarm and Stop signal for the tank interlock system of Figure 8. It is not difficult to see from Figure 9 that the formula is FALSE because the valve will remain closed if it was closed at the previous time step. Therefore, the solution of the Boolean satisfiability problem will be feasible, and the feasible solution $X_{k-1} = [\mathrm{F}]$, $X_k = [\mathrm{F}]$, $U_k = [\mathrm{FFF}]$ or transition $\mathrm{F} \xrightarrow{\mathrm{FFF}} \mathrm{F}$ will serve as a counterexample. The formula $\exists F(\tilde{X}_{1,k} \wedge \tilde{X}_{2,k})$ checks whether the horn can sound when the alarm is acknowledged by an operator for the alarm-acknowledge system of Figure 10. The formula is FALSE because the state $X = [\mathrm{TT}]$ does not exist in the state transition graph of Figure 13. Therefore, the solution of the Boolean satisfiability problem will be infeasible, hence, there is no witness.

The formulation of the Boolean satisfiability problem is illustrated using the tank interlock system of Figure 8 with the formula mentioned in the preceding section:

$$\forall G(U_{1,k} \vee U_{2,k} \rightarrow \neg \tilde{X}_{1,k}). \tag{17}$$

Rather than checking the original proposition, its negation is added to the model (Eq. 13) to formulate the following satisfiability problem:

$$\exists\left[U_{1,k-1}, U_{2,k-1}, U_{3,k-1}, U_{1,k}, U_{2,k}, U_{3,k}, \tilde{X}_{1,k-1}, \tilde{X}_{1,k}\right]$$

s.t.

$$\tilde{X}_{1,k} \leftrightarrow (\neg U_{1,k} \wedge \neg U_{2,k}) \wedge (U_{3,k} \vee \tilde{X}_{1,k})$$

$$\tilde{X}_{1,k-1} \leftrightarrow (\neg U_{1,k-1} \wedge \neg U_{2,k-1}) \wedge (U_{3,k-1} \vee \tilde{X}_{1,k-1})$$

$$(\neg U_{1,k} \wedge \neg U_{2,k}) \wedge \neg U_{3,k} \rightarrow \left(\tilde{X}_{1,k} \leftrightarrow \tilde{X}_{1,k-1}\right)$$

$$\neg(U_{1,k} \vee U_{2,k} \rightarrow \neg \tilde{X}_{1,k}), \tag{18}$$

which, if infeasible, verifies the original proposition (i.e., no state violating the original proposition exists), whereas if feasible, proves the violation of the original proposition and yields a set of feasible solutions corresponding to states and transitions that act as counterexamples.

Integer programming feasibility problem formulation

The final key step in our approach is that we exploit the equivalence between a Boolean satisfiability problem and a linear 0-1 integer programming (IP) feasibility problem (Cavalier et al., 1990) to perform the actual verification automatically and efficiently. Even though the two problems are known to be *NP-complete*, the advantages of solving the satisfiability problem as the *quantitative* IP feasibility problem have been demonstrated (Hooker, 1988). Most importantly, the IP feasibility problem derived from a satisfiability problem can be resolved by just solving the relaxed version of the problem in close to 90% of the cases (Hooker, 1988).

The Boolean satisfiability problem can be transformed automatically to an IP feasibility problem. The set of propositions in the satisfiability problem (Eq. 16) is first converted into their equivalent conjunctive normal form automatically (see the subsection on the derivation of retentive functions). Once propositions are in conjunctive normal form, it is straightforward to transform the set of propositions in Boolean variables into a set of linear inequalities in terms of binary variables. For the conjunctive normal form to be TRUE, each clause must be TRUE independently of the others and, each clause is converted into its equivalent inequality. For example, the clause $U_1 \vee \neg U_2$ is converted into $u_1 + 1 - u_2 \geq 1$, where u_1 and u_2 are binary variables corresponding to U_1 and U_2, respectively (i.e., $u_1 = 0$ and $u_2 = 1$ is not a feasible solution, but all other realizations are). Therefore, the satisfiability problem (Eq. 16) can be transformed into

$$\exists[\boldsymbol{u}_{k-1}, \boldsymbol{u}_k, \tilde{\boldsymbol{x}}_{k-1}, \tilde{\boldsymbol{x}}_k, \boldsymbol{y}_k]$$

$$\boldsymbol{A}\cdot[\boldsymbol{u}_{k-1}, \boldsymbol{u}_k, \tilde{\boldsymbol{x}}_{k-1}, \tilde{\boldsymbol{x}}_k, \boldsymbol{y}_k]^T \geq \boldsymbol{a}, \tag{19}$$

where $\boldsymbol{A}, \boldsymbol{a}$ are, respectively, an integer matrix and an integer column vector of coefficients, and $\boldsymbol{u}_{k-1}$, $\boldsymbol{u}_k \in \{0,1\}^m$, $\tilde{\boldsymbol{x}}_{k-1}$, $\tilde{\boldsymbol{x}}_k \in \{0,1\}^n$, $\boldsymbol{y}_k \in \{0,1\}^l$ are vectors of binary variables corresponding to $\boldsymbol{U}_{k-1}$, $\boldsymbol{U}_k$, $\tilde{\boldsymbol{X}}_{k-1}$, $\tilde{\boldsymbol{X}}_k$, $\boldsymbol{Y}_k$, respectively. Standard branch-and-bound algorithms are used to solve the IP feasibility problem.

Note that transformation of the Boolean satisfiability problem into its equivalence IP feasibility problem has the same computational complexity as converting Boolean formulas into conjunctive normal form, and that any propositions in the Boolean satisfiability problem can be converted efficiently into conjunctive normal forms containing a number of clauses polynomial in the number of Boolean variables as discussed in the subsection on the derivation of retentive functions.

The Boolean satisfiability problem of the tank interlock problem (Eq. 18) will be transformed into the IP feasibility problem to illustrate the procedure. The conjunctive normal form of each proposition in Eq. 18 is

$$(\neg U_{1,k} \vee \neg \tilde{X}_{1,k}) \wedge (\neg U_{2,k} \vee \neg \tilde{X}_{1,k}) \wedge (U_{1,k} \vee U_{2,k} \vee \neg U_{3,k} \vee \tilde{X}_{1,k})$$

$$(\neg U_{1,k-1} \vee \neg \tilde{X}_{1,k-1}) \wedge (\neg U_{2,k-1} \vee \neg \tilde{X}_{1,k-1}) \wedge (U_{1,k-1} \vee U_{2,k-1} \vee \neg U_{3,k-1} \vee \tilde{X}_{1,k-1})$$

$$(U_{1,k} \vee U_{2,k} \vee U_{3,k} \vee \neg \tilde{X}_{1,k} \vee \tilde{X}_{1,k-1}) \wedge (U_{1,k} \vee U_{2,k} \vee U_{3,k} \vee \tilde{X}_{1,k} \vee \neg \tilde{X}_{1,k-1})$$

$$(U_{1,k} \vee U_{2,k}) \wedge \tilde{X}_{1,k}, \tag{20}$$

which can be transformed into the IP feasibility problem:

$$\exists[u_{1,k-1}, u_{2,k-1}, u_{3,k-1}, u_{1,k}, u_{2,k}, u_{3,k}, \tilde{x}_{1,k-1}, \tilde{x}_{1,k}]$$

s.t.

$$1 - u_{1,k} + 1 - \tilde{x}_{1,k} \geq 1$$

$$1 - u_{2,k} + 1 - \tilde{x}_{1,k} \geq 1$$

$$u_{1,k} + u_{2,k} + 1 - u_{3,k} + \tilde{x}_{1,k} \geq 1$$

$$1 - u_{1,k-1} + 1 - \tilde{x}_{1,k-1} \geq 1$$

$$1 - u_{2,k-1} + 1 - \tilde{x}_{1,k-1} \geq 1$$

$$u_{1,k-1} + u_{2,k-1} + 1 - u_{3,k-1} + \tilde{x}_{1,k-1} \geq 1$$

$$u_{1,k} + u_{2,k} + u_{3,k} + 1 - \tilde{x}_{1,k} + \tilde{x}_{1,k-1} \geq 1$$

$$u_{1,k} + u_{2,k} + u_{3,k} + \tilde{x}_{1,k} + 1 - \tilde{x}_{1,k-1} \geq 1$$

$$u_{1,k} + u_{2,k} \geq 1$$

$$\tilde{x}_{1,k} \geq 1. \tag{21}$$

It takes 0.02 s to solve this IP feasibility problem using GAMS/OSL on an HP9000/735. The problem is infeasible, which means that the original specification (Eq. 7) is satisfied by the model (Eq. 13), and the tank interlock has been verified formally with respect to this constraint.

Case Study

The verification methodology proposed in this article has been applied to a number of problems from the literature (including the tank-filling interlock and alarm-acknowledge system), and from an industrial collaborator. Since all the examples in the literature are very small-scale problems in terms of the number of state variables, and the advantage of our method over existing model-checking methods lies in the problem size that can be solved without suffering from state explosion, the verification results for the examples in the literature will not be detailed here, and are summarized in Table 2. Instead, the burner management system in *North American Guidelines for Application of MOD™ 5 Burner Management* (Dow Chemical Company, 1994) will be used to illustrate the capability of our method to solve large-scale verification problems. The guidelines provide a basic burner

Table 2. Verification Results on Literature Problems

Problem	State Variables/ Total Variables	No. of IP Constraints	CPU Times (s)
Alarm acknowledge system (Moon et al. 1992)	2/12	17	0.01–0.02
Tank filling interlock	6/26	55	0.01–0.06
Furnace standard (Probst et al., 1995)	17/123	312	0.92–4.16

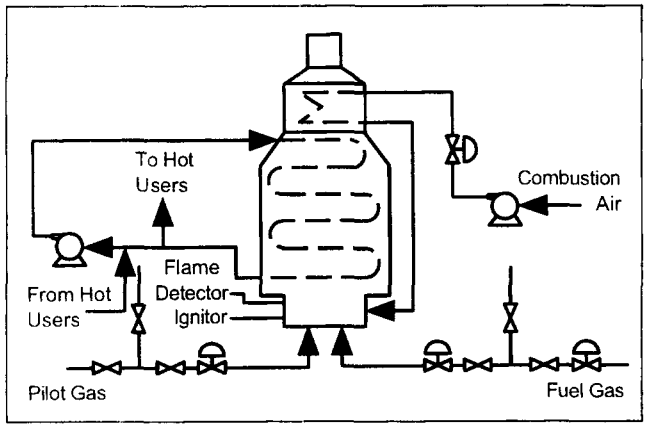

Figure 18. Fuel-gas-fired single burner.

management system for single-burner fuel-gas-fired equipment, as shown in Figure 18. The guidelines contain nine shutdown interlocks, an operation sequence involving ten steps, and seven related abort logics coupled together. Examples of each are shown in the binary logic diagrams of Figure 19. Note that they are not independent since they are coupled by sharing variables together. The system of Boolean equations is extracted from the original logic coded in DOWTRAN (a proprietary procedural control language). Table 3 shows the statistics on problem size.

Since any formal verification method will ultimately be limited by the combinatorial nature of the problem, the best way to prove the value of our approach is to apply it to a series of problems of increasing size. In order to obtain empirical results on the performance of our approach, we formulate a global burner management system by combining a number of single-burner management systems in parallel and by adding a global shutdown logic, as shown in Figure 20. Then, the global model is tested against the five specifications listed in Table 4 by increasing the number of single burners from 1 to 10. All the IP feasibility problems are solved by standard branch-and-bound codes in GAMS/OSL (Brooke et al., 1992). All the IP problems tested are infeasible, meaning that all the specifications tested are satisfied by the model.

Table 5 and Figure 21 show model statistics. Note that the problem size in terms of the number of constraints and total number of variables is increasing linearly with the number of state variables, which demonstrates that the size of the model that can be constructed is not limited by the amount of computer memory available due to its implicitness. Therefore, very large-scale verification problems can be formulated in our approach, which is impossible in explicit enumeration-based model checking.

Table 6 shows the verification performance in terms of the number of state variables. First of all, the largest verification problem has 441 state variables and 652 inputs, resulting in

Table 3. Model Statistics for Single-Burner Management System

Category	No.
No. of propositions	274
No. of inputs	67
No. of state variables	44
No. of outputs	20

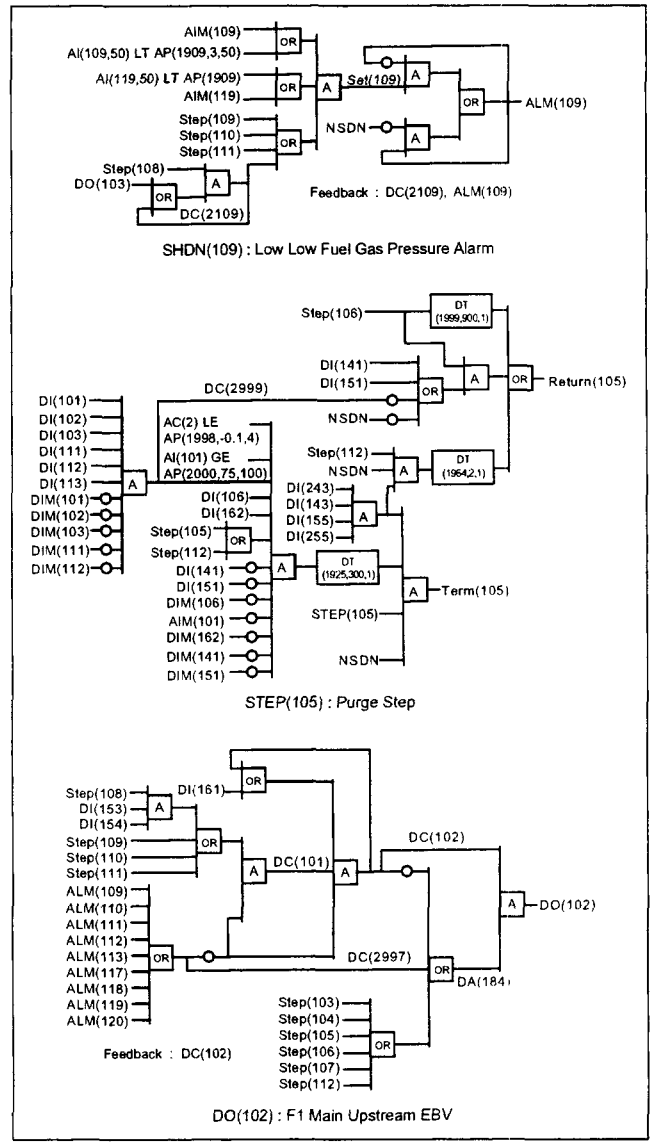

Figure 19. Sample logic of single-burner management system.

the possibility of more than 10^{132} states and 10^{474} transitions. Verification took from 3.5 to 7.5 min of CPU time on an HP9000/735, depending upon the specifications tested. Note that the IP feasibility problem was resolved by just solving the relaxed linear-programming problem in several cases, and in other cases, the number of nodes visited during branch-

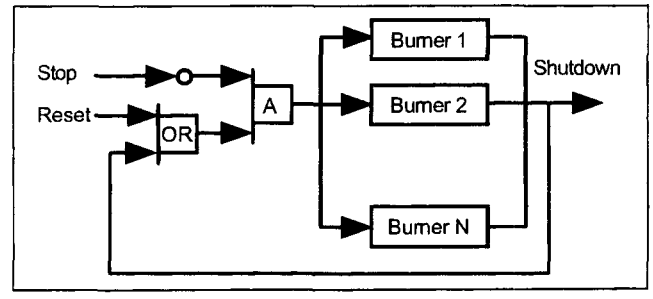

Figure 20. Global-burner management system.

Table 4. Specifications Verified

No.	Informal Specification	Formal Specification
1	STOP signal should bring all burners in safe shutdown states, and should override RESET signal	$\forall G[\text{STOP} \wedge \text{RESET} \rightarrow \text{STEP}_1(112) \wedge \cdots \wedge \text{STEP}_N(112)]$
2	Once STOP signal becomes ON, shutdown signal should remain ON even though STOP signal becomes OFF	$\forall G[\neg \text{STOP} \wedge \text{SHUTDOWN} \rightarrow \forall N(\text{SHUTDOWN})]$
3	Upon receiving STOP signal, outputs of all main fuel downstream block valves should be aborted	$\forall G[\text{STOP} \rightarrow \neg(\text{DO}_1(103) \vee \cdots \vee \text{DO}_N(103))]$
4	Upon receiving STOP signal, outputs of all pilot gas upstream block valves should be aborted	$\forall G[\text{STOP} \rightarrow \neg(\text{DO}_1(112) \vee \cdots \vee \text{DO}_N(112))]$
5	Upon receiving STOP signal, outputs of all burner electronic ignitors should be aborted	$\forall G[\text{STOP} \rightarrow \neg(\text{DO}_1(114) \vee \cdots \vee \text{DO}_N(114))]$

Table 5. Model Statistics for Global Burner Management System

Unit	n	m	Total	Constraints	States	Transitions
1	45	67	169	1,116	3.5×10^{13}	5.2×10^{33}
2	89	132	335	2,220	6.2×10^{26}	3.4×10^{66}
4	177	262	667	4,428	1.9×10^{53}	1.4×10^{132}
6	265	392	999	6,636	5.9×10^{79}	6.0×10^{197}
8	353	522	1,331	8,844	1.8×10^{106}	2.5×10^{263}
10	441	652	1,663	11,052	5.7×10^{132}	4.8×10^{474}

Unit: number of single burners in the global burner management system; n: number of state variables; m: number of inputs; Total: total number of variables (inputs, outputs, state variables, internal variables) (Note that internal variables are intermediate variables introduced during model construction); Constraints: number of inequality constraints generated; States: upper bound on the number of possible states; Transitions: upper bound on the number of possible transitions.

and-bound search for IP solution is very small (less than 300). Figure 22 shows how the verification time depends on the problem size in terms of the number of state variables on the log-log scale. On a log-log scale plot, the polynomially growing function $y = x^n$ appears as a straight line with a slope n. Note that in all cases, the verification time is growing roughly quadratically (i.e., $n \approx 2$) with respect to the number of state variables.

Conclusions

We have shown that formal verification of large-scale sequential logic systems can be conducted efficiently without encountering the state explosion problem. The key new result that facilitates this is the development of a novel implicit representation for sequential logic systems. In addition, significant computational benefits arise from solving the verification problem in the domain of binary variables rather than in the domain of Boolean variables.

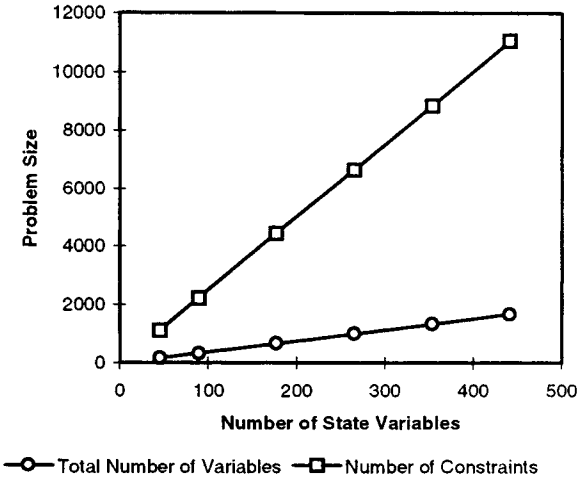

Figure 21. Problem size vs. number of state variables.

Sequential logic systems are represented implicitly by a Boolean state-space model that embed all possible states and transitions in a compact closed form. Specifications are represented in a subset of temporal logic. Combining the model with specifications, the verification problem is formulated as a Boolean satisfiability problem, which in turn is transformed into its equivalent IP feasibility problem. The IP feasibility problem is solved using a standard branch-and-bound algorithm, whose solution determines whether the specification is satisfied or not, yielding counterexamples or witnesses, as necessary. Note that all the intermediate steps involved can be automated and can be performed efficiently in polynomial time.

In principle, the verification problem is combinatorial. The key feature of our approach is to confine the combinatorial nature of the problem to the solution of the IP feasibility problem, which implies that (1) model formulation is not combinatorial at all, and (2) the whole verification problem can be solved efficiently if we can solve the resulting IP feasibility problems efficiently. Empirical studies indicate that the IP feasibility problem can be solved very efficiently in polynomial time using a standard branch-and-bound algorithm even though the IP feasibility problem is in the worst-case combinatorial. This high efficiency in terms of computational cost is because the IP feasibility problem was solved by just solving the relaxed linear programming problem in many cases,

Table 6. Verification Results

	Specification 1		Specification 2		Specification 3		Specification 4		Specification 5	
n	Nodes	CPU	Nodes	CPU	Nodes	CPU	Nodes	CPU	Nodes	CPU
45	0	1.56	10	2.46	0	2.45	0	2.67	0	1.64
89	0	7.72	6	5.25	21	11.46	0	8.22	21	8.63
177	41	38.48	21	32.45	41	43.21	41	35.60	21	24.47
265	61	67.81	41	66.06	61	62.19	41	73.38	41	70.46
353	61	122.34	61	146.31	61	138.92	81	101.73	81	163.58
441	141	296.44	84	211.51	241	447.40	81	211.83	81	235.60

n: number of state variables; Nodes: number of binary nodes visited during branch-and-bound search in GAMS/OSL; CPU: CPU times in seconds using GAMS/OSL in HP9000/735.

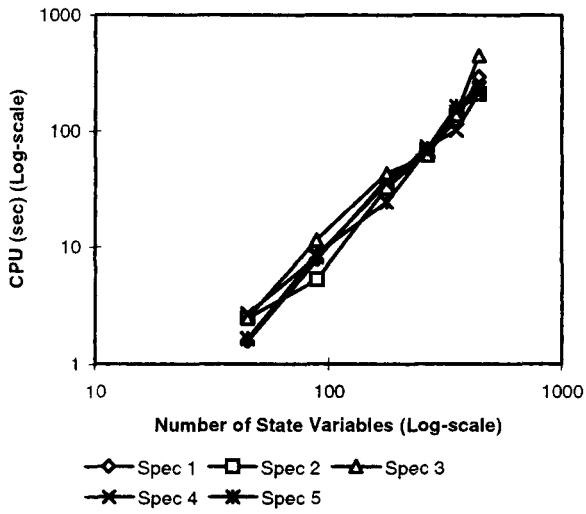

Figure 22. Verification times vs. problem size.

and in other cases, the number of nodes visited during a branch-and-bound search for IP solution is very small. We have no theoretical explanation for this observation yet; however, it has been noted before that this is due to the special structure of the IP problem, and arises from the use of the conjunctive normal form (Cavalier et al., 1990). Here, we observe that the implicit enumeration conducted by a branch-and-bound algorithm based on information from partial relaxations of the IP feasibility problem is dramatically more efficient and tractable than enumeration-based verification techniques.

Literature Cited

AIChE/CCPS, *Guidelines for Safe Automation of Chemical Processes*, AIChE/CCPS, New York (1993).

Alur, R., C. Courcoubetis, T. A. Henzinger, and P. Ho, "Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems," *Hybrid Systems*, Lecture Notes in Computer Science, **736**, Springer-Verlag, New York (1993).

ANSI/ISA, *Binary Logic Diagrams for Process Operations*, ANSI/ISA-S5.2-1976 (R1981), ISA, New York (1981).

Barton, P. I., and T. Park, "Analysis and Control of Combined Discrete/Continuous Systems: Progress and Challenges in the Chemical Processing Industries," *AIChE Symp. Ser.* (1997).

Bell, R., "Safety in Chemical Process Automation: HSE Approach," *Int. Symp. and Workshop on Safe Chemical Process Automation*, AIChE/CCPS, New York (1994).

Bose, S., and A. Fisher, "Automatic Verification of Synchronous Circuits using Symbolic Logic Simulation and Temporal Logic," *Proc. IMEC-IFIP Int. Workshop on Applied Formal Methods for Correct VLSI Design*, Leuven, Belgium (1989).

Boy de la Tour, T., "Minimizing the Number of Clauses by Renaming," *Proc. Int. Conf. on Automated Deduction*, Kaiserslautern, France (1990).

Brooke, A., D. Kendrick, and A. Meeraus, *GAMS Release 2.25 A User's Guide*, Scientific Press, San Francisco (1992).

Bryant, R. E., "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans Comput.*, **C-35**(8), 677 (1986).

Burch, J. R., E. M. Clarke, and D. E. Long, "Representing Circuits more Efficiently in Symbolic Model Checking," *Proc. ACM/IEEE Design Automation Conf.* (1991).

Burch, J. R., E. M. Clarke, D. E. Long, K. L. McMilan, and D. L. Dill, "Symbolic Model Checking for Sequential Circuit Verification," *IEEE Trans. Comp.-Aided Des. Integrated Circuits Syst.*, **13**(4), 401 (1994).

Burch, J. R., E. M. Clarke, K. L. McMillan, and D. L. Dill, "Sequential Circuit Verification Using Symbolic Model Checking," *Proc. ACM/IEEE Design Automation Conf.* (1990).

Cavalier, T. M., P. M. Pardalos, and A. L. Soyster, "Modeling and Integer Programming Techniques Applied to Propositional Calculus," *Comput. Oper. Res.*, **17**(6), 561 (1990).

Clarke, E., and E. A. Emerson, "Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic," *Proc. Workshop on Logics of Programs*, Lecture Notes in Computer Science, **132**, Springer-Verlag, New York (1981).

Clarke, E., E. A. Emerson, and A. P. Sistla, "Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications," *ACM Trans. Program. Lang. Syst.*, **8**(2), 244 (1986).

Clarke, E., O. Grumberg, and D. Long, "Verification Tools for Finite-State Concurrent Systems," *A Decade of Concurrency: Reflections and Perspectives, Proc. REX School/Symp.*, Noordwijkerhout, The Netherlands (1993).

Clockskin, W. F., and C. S. Mellish, *Programming in Prolog*, 2nd ed., Springer-Verlag, New York (1984).

Coudert, O., J. C. Madre, and C. Berthet, "Verifying Temporal Properties of Sequential Machines without Building their State Diagrams," *Proc. Workshop on Computer-Aided Verification* (1990).

Dow Chemical Company, *North American Guidelines for Application of MOD(TM) 5 Burner Management* (1994).

Emerson, E. A., "Temporal and Modal Logic," *Handbook of Theoretical Computer Science*, Elsevier, Amsterdam (1990).

Friedman, A. D., *Fundamentals of Logic Design and Switching Theory*, Computer Science Press, Rockville, MD (1986).

Hähnle, R., "Short CNF in Finitely-Valued Logics," *Proc. Int. Symp. on Methodologies for Intelligent Systems*, Trondheim, Norway (1993).

Halbwachs, N., D. Pilaud, and F. Ouabdesselam, "Specifying, Programming and Verifying Real-time Systems Using a Synchronous Declarative Language," *Proc. Int. Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble (1989).

Hooker, J. N., "Generalized Resolution and Cutting Planes," *Ann. Oper. Res.*, **12**, 217 (1988).

Jagadeesan, L. J., C. Puchol, and J. E. Von Olnhausen, "Safety Property Verification of ESTEREL Programs and Applications to Telecommunications Software," *Proc. Int. Conf. Computer Aided Verification*, Liège, Belgium (1995).

Jeong, S. H., K. W. Lee, and I. Moon, "Symbolic Safety Analysis of Automated Fuel Cell Processes," AIChE Meeting, Miami, FL (1995).

Kestne, Y., A. Pnueli, J. Sifakis, and S. Yovine, "Integration Graphs: A Class of Decidable Hybrid Systems," *Hybrid Systems*, Lecture Notes in Computer Science, **736**, Springer-Verlag, New York (1993).

Moon, I., "Modeling Programmable Logic Controllers for Logic Verification," *IEEE Contr. Syst. Mag.*, **14**(2), 53 (1994).

Moon, I., G. J. Powers, J. R. Burch, and E. M. Clarke, "Automatic Verification of Sequential Control Systems Using Temporal Logic," *AIChE J.*, **38**(1), 67 (1992).

Nemhauser, G. L., and L. A. Wolsey, *Integer and Combinatorial Optimization* Wiley, New York (1988).

Otter, J. D., *Programmable Logic Controllers: Operation, Interfacing and Programming*, Prentice Hall, Englewood Cliffs, NJ (1988).

Park, T., and P. I. Barton, "Binary Logic Diagrams Imply Nonunique Functionality," *ISA Trans.*, **35**, 337 (1996).

Plaisted, D. A., and S. Greenbaum, "A Structure-Preserving Clause form Translation," *J. Symb. Comput.*, **2**, 293 (1986).

Probst, S. T., and G. J. Powers, "Automatic Verification of Chemical Processes in the Presence of Failure Modes," AIChE Meeting, San Francisco (1994).

Probst, S. T., G. J. Powers, D. E. Long, and I. Moon, "Verification of a Logically Controlled Solids Transport System Using Symbolic Model Checking," *Comput. Chem. Eng.*, **21**(4), 417 (1997).

Probst, S. T., A. L. Turk, and G. J. Powers, "Formal Verification of a Furnace System Standard," AIChE Meeting, Miami, FL (1995).

Victor, J., "Designing Safe Interlock Systems," *Chem. Eng.* (1979).

Manuscript received Dec. 3, 1996, and revision received May 8, 1997.