Automatic Verification of Sequential Control Systems Using Temporal Logic

Il Moon and Gary J. Powers
Dept. of Chemical Engineering

Jerry R. Burch and Edmund M. Clarke
Dept. of Computer Science

Carnegie Mellon University, Pittsburgh, PA 15213

Clarke et al. (1986) have developed a model-based verification method and have applied it to validation of VLSI circuits. We have used the method to test automatically the safety and operability of discrete chemical process control systems. The technique involves: 1) a "system model" describing the process and its software; 2) "assertions" in temporal logic expressing user-supplied questions about the system behavior with respect to safety and operability; and 3) a "model checker" that determines if the system model satisfies each of the assertions and provides a counterexample to locate the error if one exists. Temporal logic is used for reasoning about occurrence of events over time. To reveal discrete event errors, we have applied the verification method to a simple combustion system and an alarm acknowledge system.

Introduction

As chemical processes increasingly use computers for their control, avoiding computer control failures becomes more important. This article describes a verification method that automatically determines the safety and operability of sequential discrete chemical process control systems.

Current methods for assessing the safety and reliability of chemical processes are diverse (Dhillon, 1988). Hazard and operability studies (HAZOPs) are widely used for identifying hazards or operability problems. This method involves preparing a list of all possible deviations from normal operating conditions and how the deviations might occur. The consequences on the process are assessed, and the means available to detect and correct the deviations are examined. This method is difficult to apply to complex systems because of the large number of failure combinations and the many interactions between components and subsystems. Fault tree analysis (FTA) overcomes some of the limitations of HAZOP. The critical difference between the two methods is the direction in which the analysis is performed. HAZOP involves generation of event sequences from initiating events to final events, while FTA begins with the final event and works backward to initiating events. In FTA, only event sequences leading to failures of interest are considered. The fault tree method can be quantified where data are available on failure probabilities of primal events. These methods are systematic approaches for determining if failures or changes in the process equipment or procedures will result in undesirable process events such as fatalities, injuries, environmental damage, or unintended process shutdown.

Checking interactions between process equipment and computer software using these techniques presents numerous problems. First, the complexity of the control system hardware, operating system and application code causes large combinatorial search problems for HAZOP and FTA types of risk assessment. Second, sequential or batch processing systems introduce additional complexity due to the large number of possible states that the process and its control system might attain. Finally, models for predicting software integrity are commonly restricted to the software code itself without including the process or system that is controlled by the code. Excluding the chemical process in the assessment of the software precludes the detection of errors that are caused by the misapplication of the code or by failures that propagate from the process through the code. These types of software appli-

Correspondence concerning this article should be addressed to G. J. Powers.

cation errors often are difficult to detect and can cause large consequence failures in the processing plant.

This article addresses the use of an automatic formal verification method for process control systems that involve discrete event dynamics. The objective of the study is to determine whether a formal verification method would be able to identify effectively errors in a system that includes both the control system and the chemical process equipment. The method requires two inputs:

- A state transition model for the system to be verified including software and process units manipulated by a sequential controller such as the programmable logic controller (PLC).
- A list of questions about the possible behavior of the system.

Other less formal approaches have been used in several industries. These techniques normally rely on testing the functionality of the control software at each stage of its development. Field testing prior to commissioning of the control system is a critical activity that commonly involves checking the control system while changing one input at a time, followed by larger-scale functional tests with inert material in the process.

Several recent studies have given guidelines for the safety and reliability of computer-based process control systems. The Pharmaceutical Manufacturers Association's Computer System Validation Committee has addressed several factors to be considered for validating automated process controls in bulk pharmaceutical operations (Chapman, 1989). Shaw (1991) has given a checklist to reduce human errors in distributed control systems. The Institute of Electrical and Electronics Engineers has provided manual guidelines for software quality assurance plans (IEEE, 1984) and software unit testing (IEEE, 1987). The methods in these guidelines are mostly manual and suffer from combinatorial search problems and a potential lack of completeness. Verification and validation methods through the software life cycle were summarized by Wallace (1989). These methods have been used for software testing in aerospace applications and computer system development. These pioneering efforts have indicated that organized search for software and control system errors can greatly improve system integrity, although it is time-consuming. In this work, some formal verification techniques are combined with chemical engineering models for processing systems to verify discrete event chemical processes.

Engineers use discrete event control systems widely for on/off sequential control of batch systems, alarms, interlocks, and startup and shutdown procedures. The search for safety errors in discrete chemical process control systems depends on models of the systems. Ho (1989) classified the models of discrete event dynamic systems by logical, algebraic and stochastic models for both timed and untimed systems. Yamalidou (1990) examined the behavior of discrete chemical processing plants using Petri nets (untimed logical model), Minimax algebra (timed algebraic model), and temporal logic (timed logical model). Simulation, using these discrete models, is one way of investigating the behavior of a sequential chemical process. We could use repeated simulations to detect possible errors in the control system. While effective for small problems, these simulation approaches commonly have combinatorial problems when applied to complex systems. We use a finite state machine model and temporal logic for reasoning about occurrence of events over time.

Model checking using temporal logic has the potential for reducing the growth rate of the search space. The application to the verification of VLSI circuit designs and communication protocols containing up to 10^{20} states has shown the power of this method (Burch, 1990). We have combined state transition models of chemical process equipment with similar models of the control system and its software, and developed a search method using temporal logic for finding errors in the control system.

The method identifies errors by comparing a question about system behavior with the model of the system. The question is given as a series of temporal logic statements. For example, an informal question in furnace operation, "Is there any future situation in which fuel flows without being burned?" may be expressed using temporal logic as "*EF*(fuel AND (NOT flame))" where *EF*(*p*) is a temporal logic formula meaning "there Exists a state in the Future where *p* holds." The next step is to determine whether the model of the system satisfies each question.

We applied the model checking method of Clarke et al. (1986) to test chemical process control systems. The two case studies used are: a combustion system involving an air/fuel burner, a flame detector, and shutdown and startup procedures, and an alarm system using PLC software.

The results of these studies indicate that the model checking approach can identify errors in discrete chemical process control systems. The main advantages of this method for testing discrete event systems are:

- Process models are included so that the interactions between software and process hardware are tested.
- An algorithmic search is used to make the verification process more complete.
- Alternate designs can be compared by testing them with the same set of assertions.
- The search method is automated and has the potential for testing complex systems.

Model Checking Verification

An overview of the verification method is shown in Figure 1, where the system description and the assertions are inputs to the model checker. The *system description* is a state transition model of the system to be tested. The model is derived from the process flow diagram, control software, and piping and instrumenting diagram (P&ID). *Assertions* are questions associated with safety and operability coming from industrial standard checklists, process design specifications, or other methods like HAZOP or fault tree analysis. Assertions are

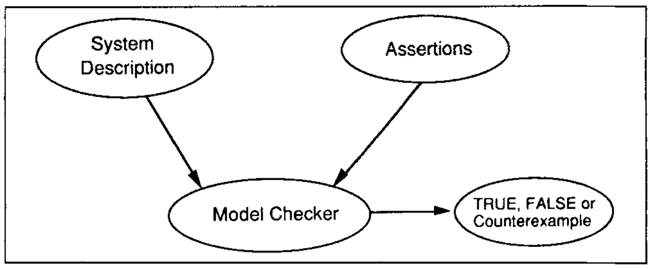

Figure 1. Overview of the model checking verification method.

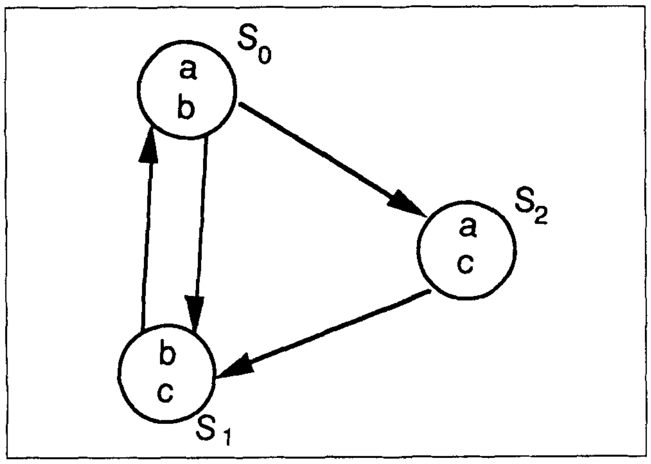

Figure 2. State transition graph.

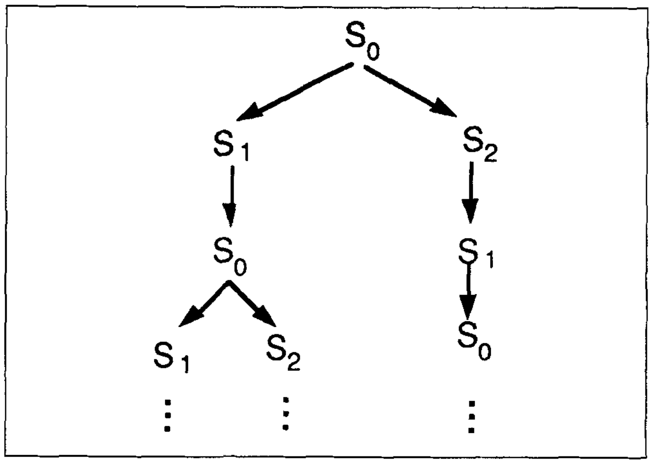

Figure 3. Corresponding computation tree to Figure 2 with initial state s_0

expressed in temporal logic. The *model checker* searches the state space of the system and determines the truth of the assertions.

The next section describes the modeling of sequential systems and is followed by the model checking algorithm and a description of how to express assertions.

System Modeling

The behavior of the process equipment, the operating procedures and the process control software in the form of a PLC ladder diagram is converted into a labeled state transition graph. Figure 2 shows an example of a state transition graph, where a circle indicates a state, s_i, and an arrow denotes a state transition. The state variables (called atomic propositions) of the system can take the values of TRUE or FALSE, which represent the discrete state values of on-off valves, pumps, relays, tank levels, switches, and so on. Only the variables that have the value TRUE in a given state appear in the circle representing that state. The arrows in the state transition graph represent the transition of the system from one state to another. The immediate successors of a state s are the states that can be reached from s in one step. More formally, a state transition structure $M=(S, R, P)$ includes three components where

- S is a finite set of states.
- R is a binary relation on S which gives the possible transitions between states.
- $P(s)$ is the set of TRUE atomic propositions in state s, where an atomic proposition is the state variable that denotes the property of interest and has either a TRUE or FALSE value.

Using a library of models of process equipment, operating procedures and the PLC software, a sequential system is modeled as a state transition structure. The following section illustrates how to express and check assertions in this method.

Computation Tree Logic

Computation tree logic (CTL) formulas are used to express assertions about the system being verified. These assertions can be provided by the system analyst, standard system specifications, or error types discovered in previous designs, and are used to detect operability, reliability and safety features. The CTL model checker program automatically tests whether the assertions are TRUE of the system model.

The truth of a CTL formula is relative to a state transition graph. To understand how the truth of a formula depends on a state transition graph, it is helpful to think of unwinding the graph into an infinite tree with the initial state as the root. The tree obtained in this manner is called a computation tree. Paths in the tree represent possible behaviors of the system modeled by the state transition graph. For example, Figure 3 presents the tree corresponding to Figure 2 with the initial state s_0. One of the paths in the tree is s_0, s_1, s_0, s_2, and so on.

The simplest CTL formulas consist of just an atomic proposition. If p is an atomic proposition, then the formula p is TRUE of a state s if and only if (iff) p labels s: that is, p is an element of $P(s)$. Formulas can be built up using the standard operators of *negation* (written $\sim$), *and* (written &), *or* (written |), and *imply* (written $\rightarrow$). CTL is distinguished from elementary propositional logic by the modal operators AX, EX, AU and EU, where A (for all computation paths) and E (for some computation path) are path quantifiers, and X (next time) and U (until) are state quantifiers. With these operators, it is possible to construct formulas whose truth in a state s depends on the labeling of states other than s. Thus, one can construct formulas that assert restrictions on the kinds of behaviors that can start in a given state.

For example, $EX(f)$ is TRUE of a state s iff f is TRUE of some immediate successor of s; $AX(f)$ is TRUE iff f is TRUE of all immediate successors. The formula $E[f1\ U f2]$ is TRUE of a state s iff there exists a path starting at s with the following property: there exists an initial prefix of the path such that $f2$ holds at the last state of the prefix, and $f1$ holds at all other states along the prefix. The formula $A[f1\ U f2]$ makes the analogous assertion about all of the paths starting with state s.

In summary, the formal syntax for CTL is such that

- Every atomic proposition $p \in AP$ is a CTL formula.
- If $f1$ and $f2$ are CTL formulas, then so are $\sim f1$, $f1 \& f2$, $AX(f2)$, $EX(f1)$, $A[f1\ U f2]$ and $E[f1\ U f2]$.

The following abbreviations are used in writing CTL formulas:

$EF(f) \equiv E[\text{TRUE}\ U f]$ means that there is some path from s_0 that leads to f: that is, f holds *potentially*.

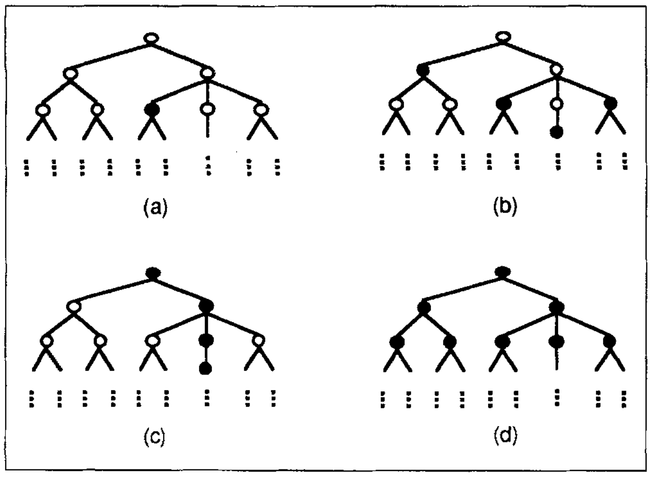

Figure 4. CTL operators.

● $=p$, ○ $= \sim p$ (*a*) $EF(p)$: p potentially holds (*b*) $AF(p)$: p is inevitable (*c*) $EG(p)$: p holds at every state in some path (*d*) $AG(p)$: p is invariant.

$AF(f) \equiv A$[TRUE U f] means that f holds in the future along every path from the initial state s_0: that is, f is *inevitable*.

$EG(f) \equiv \sim AF(\sim f)$ means that there is some path from s_0 on which f holds at every state.

$AG(f) \equiv \sim EF(\sim f)$ means that f holds at every state on every path from s_0: that is, f holds *globally*.

Figure 4 shows how simple correctness properties can be represented using these operators, where the black circle and the white circle indicate that the atomic proposition p is TRUE and FALSE, respectively, in the corresponding states. More complex formulas can be represented by combining the CTL operators. For example, $AG\ AF\ (f)$ means that for all states s, all paths starting from s contain at least one state where f is TRUE. This is the same as saying that f is TRUE infinitely often on all paths starting from the initial state. The expression $EF\ EG\ (f)$ means that at some state in the future there exists a path along which f is TRUE at every state.

The model checker automatically tests whether an assertion is satisfied in the system model. The algorithm processes a formula bottom up, checking the shortest subformulas before the subformulas that contain them. For each CTL operator, there is an algorithm for determining the truth of a formula constructed with the operator, given that the truth of the subformulas has already been determined. The model checker is the combination of these algorithms, together with an algorithm for producing a counterexample trace in response to a FALSE formula. The counterexample trace is a sequence of states that demonstrates why the formula is FALSE. This feature is quite useful for locating the cause of errors in the system being verified. A more thorough description of the model checker program is given by Clarke et al. (1986, 1987).

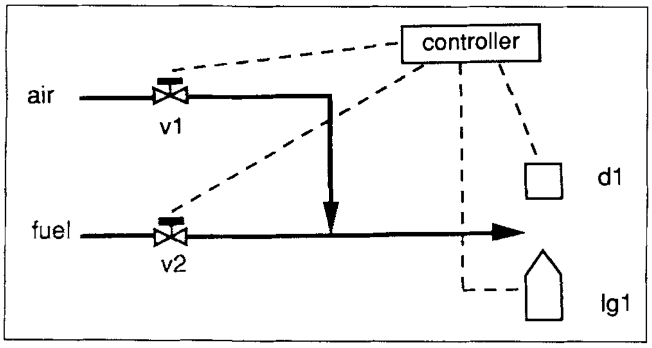

Figure 5. Combustion system.

```
1. Start with initial condition
2. Open the air valve (v1=1)
3. Open the fuel valve (v2=1)
4. Turn on the ignitor (Ig1=1)
5. Turn off the ignitor (Ig1=0)
6. If there exists flame (d1=1),
       then go to step 7
       else go to step 4
7. If shutdown button is on or flame disappears,
       then stop
       else go to step 7
```

Figure 6. Operating sequence No. 1 for a combustion system.

The following two examples illustrate the method, and demonstrate its usefulness in the verification of discrete chemical process control systems. The first is a combustion system at the design level, and the second is an alarm acknowledge system at the detailed software level.

Case Studies

A combustion system

This example illustrates the model checking verification method by testing a chemical process using a state transition graph and CTL assertions. Figure 5 shows a combustion system, where $v1$ and $v2$ are normally closed solenoid valves, $d1$ is a flame detector, and $Ig1$ is an ignitor. Assume that a designer has proposed one operating sequence as shown in Figure 6. The goal of this analysis is to detect a safety error in the operating sequence before the designer implements the corresponding control system. A combined model of the valves, the detector, the ignitor and the operating sequence is presented by the state transition graph shown in Figure 7. This graph is the input to the model checker as the system description.

A trace of the CTL model checker applied to the system description is shown in Figure 8, where lines in bold face represent inputs from the user. The first test assertion is:

$$EF(\text{air \& fuel \& flame})$$

in other words, "Is it TRUE that there exists a state in the future where air, fuel, and flame coexist?" as shown in line 4. This test determines whether a situation arises where air, fuel, and flame are present at the same time in the model. The answer is TRUE as shown in line 5: that is, such a situation *can occur*, so that the burner works in at least one case as specified by the assertion.

Now let us check if a steady, unsafe state exists in which fuel flows without flame. The assertion for testing this condition is:

$$EF\ EG\ (\text{fuel \& } \sim \text{flame})$$

in other words, "Is there any state that begins a path where

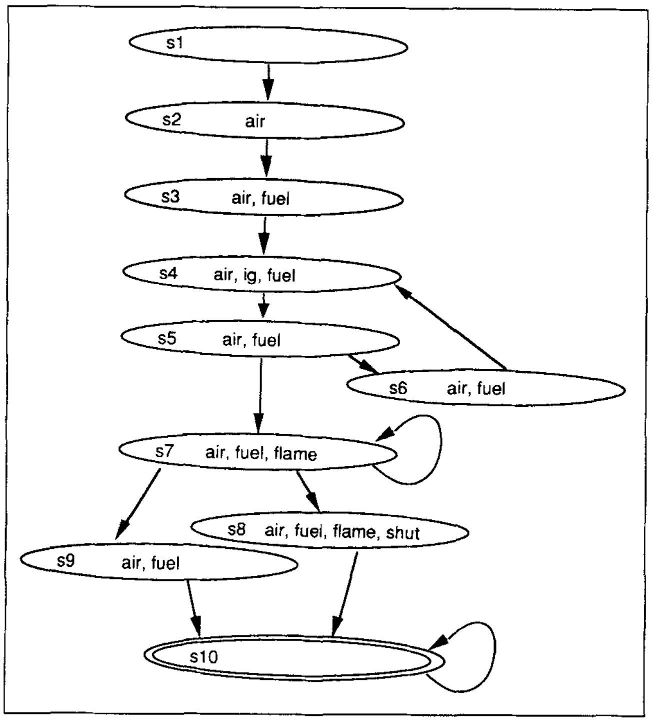

Figure 7. State transition graph for the operating sequence No. 1.

fuel exists and flame does not exist all along the path?" The negation of this assertion is used to get a counterexample as shown in line 7 of Figure 8. The answer to this test is FALSE because a potentially unsafe infinite loop exists at states 4, 5, 6, 4, 5, 6, and so on. The location of the loop is determined automatically by the model checker and displayed in lines 21 through 26. The method for locating the loop is based on proposing the counterexample given in line 10. The continuation of the counterexample development for line 10 is given in lines 16 through 19. Hence, the CTL model checker shows that the operating sequence No. 1 implies a potentially unsafe condition. This path is for the condition where the fuel will not ignite and operating sequence No. 1 continues to turn on the ignitor in an attempt to achieve ignition. The location of the unsafe condition might suggest process or software changes. The current methods do not automatically revise the system design.

```
 1        CTL MODEL CHECKER
 2  Taking input from high_op1.emc...
 3
 4  |= EF (air & fuel & flame).
 5  The assertion is TRUE.
 6
 7  |= ~EF EG (fuel & ~flame).
 8  The assertion is FALSE.
 9
10  EF EG (fuel & ~flame)
11        is true in state 1 because of the path:
12  State 1:
13  State 2:  air
14  State 3:  air fuel
15
16  AF ~(fuel & ~flame)
17        is false in state 3 because
18  EG ~~(fuel & ~flame)
19         is true in state 3.
20  An example of such a path is:
21  State 3:  air fuel
22  State 4:  air fuel ig
23  State 5:  air fuel
24  State 6:  air fuel
25  State 4:  air fuel ig
26        ...
```

Figure 8. CTL model checker execution for the operating sequence No. 1.

```
1. Start with initial condition
2. Open the air valve (v1=1)
3. Turn on the ignitor (Ig1=1)
4. Open the fuel valve (v2=1)
5. If there exists flame (d1=1),
       then go to step 6
       else turn off the ignitor (Ig1=0) and
            close the fuel valve (v2=0) and
            go to step 3
6. Turn off the ignitor (Ig1=0)
7. If shutdown button is on or flame disappears,
       then stop
       else go to step 7
```

Figure 9. Operating sequence No. 2 for a combustion system.

Let us consider another design proposal, operating sequence No. 2 as shown in Figure 9. This procedure uses a different sequence based on detecting the flame to control the ignition of the fuel. The corresponding state transition graph is shown

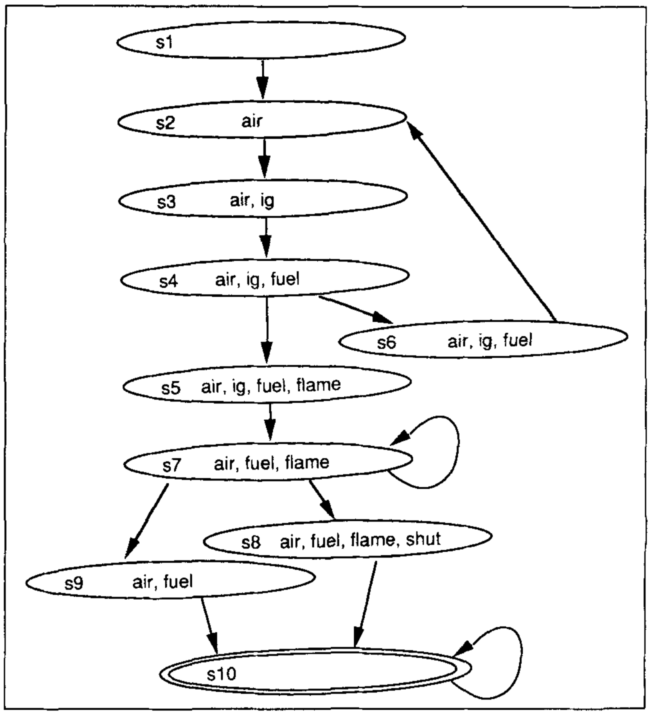

Figure 10. State transition graph for the operating sequence No. 2.

```
1        CTL MODEL CHECKER
2    Taking input from high_op2.emc...
3
4    |= EF (air & fuel & flame).
5    The assertion is TRUE.
6
7    |= ~EF EG (fuel & ~flame).
8    The assertion is TRUE.
```

Figure 11. CTL model checker execution for the operating sequence No. 2.

in Figure 10. The answer to the safety question ~*EF EG* (fuel & ~ flame) for this sequence is TRUE as shown in the execution file, Figure 11 in lines 7 and 8. The unsafe path found in operating sequence No. 1 is not present in the revised system No. 2.

This combustion system example illustrates that once a state transition graph (system description) and assertions are defined, then the CTL model checker can be used to automatically determine the truth of the assertions in the model. Other assertions about system operability and reliability could be tested in the same manner. The next example includes another application of this verification method to test the correctness of software used in a process control alarm system.

An alarm acknowledge system

This example uses a programmable logic controller (PLC) to demonstrate an application of the verification method at the software level. PLCs are used extensively in chemical industry in a wide variety of applications. Programming languages for PLCs include ladder diagrams, Boolean expressions, and Grafcet. Among these languages, the ladder diagram is the most popular. Laduzinsky (1990) indicated that 70% of the PLC programmers preferred this language.

Consider the alarm system shown in Figure 12, where the high-level and the high-temperature alarms of a storage tank are activated by a PLC, and the horn is acknowledged by an operator. A ladder diagram used for the PLC is shown in Figure 13. The vertical rails of the diagram indicate the power source and sink, while the horizontal lines, called rungs, indicate the possible current (or signal) flow. Various symbols for buttons, contacts, and coils can be placed on the rungs of the ladder. If the appropriate contacts are activated, a coil is energized and its associate relays are closed if they are normally open relays. For example, closing the high-level contact (HiL) in rung 1 activates the relay coil L1 in rung 1 and causes closing of the relay L1 in rung 3.

The ladder diagram in Figure 13 includes one pushbutton (PB), one horn, and two sensors (high level and high temperature). The horn and the acknowledge relay (Ack) are latched in rung 3 and rung 4, respectively: that is, once a value is changed, the value is retained. The contact Ack in rung 3 is normally closed, and other contacts are normally open. The

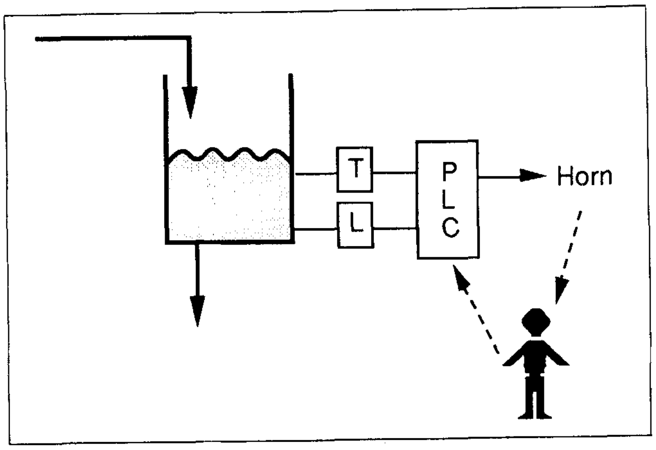

Figure 12. Alarm acknowledge system.

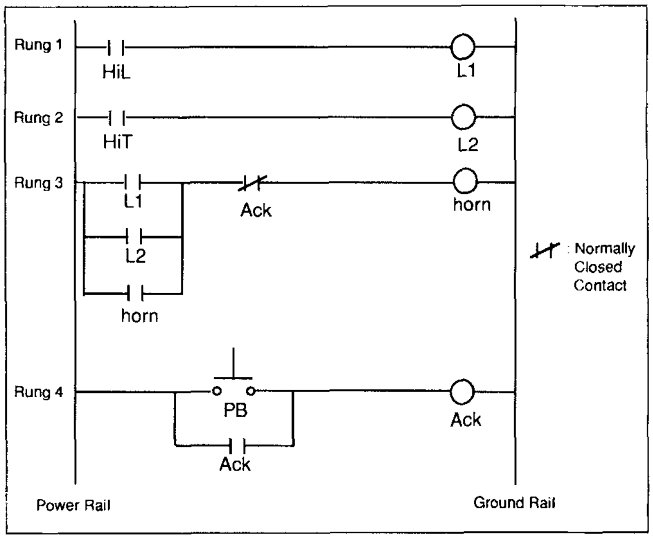

Figure 13. Ladder diagram for the alarm acknowledge system.

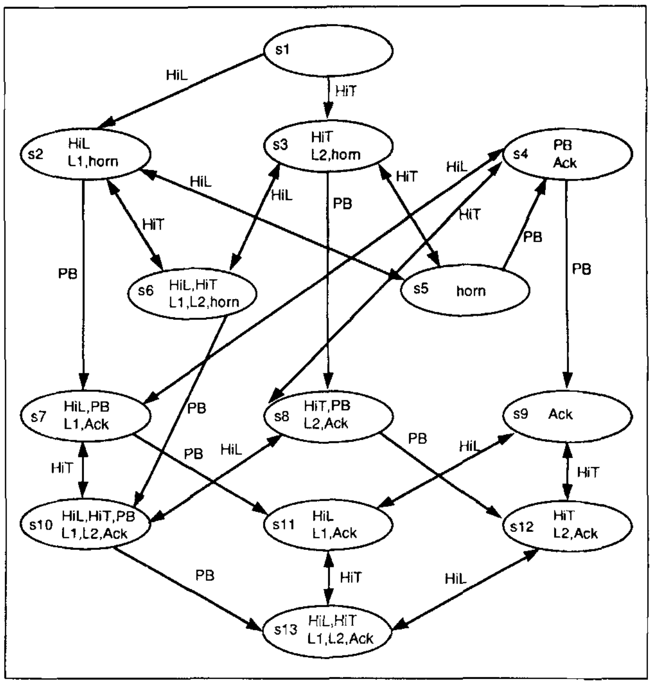

Figure 14. State transition graph for the alarm acknowledge system.

```
1      CTL MODEL CHECKER
2   Taking input from alarm1.emc...
3
4   |= AG(HiL -> AF horn).
5   The assertion is FALSE.
6
7   EF ~(HiL -> AF horn)
8          is true in state 1 because of the path:
9   State 1:
10  State 2:  HiL L1 horn
11  State 7:  HiL L1 PB Ack
12
13  HiL -> AF horn
14         is false in state 7 if:
15         1) ~HiL
16                is false in state 7, AND
17         2) AF horn
18                is false in state 7.
19
20  ~HiL
21         is false in state 7 because the following propositions are true:
22  HiL
23
24  AF horn
25         is false in state 7 because
26  EG ~horn
27         is true in state 7.
28  An example of such a path is:
29  State 4:  PB Ack
30  State 7:  HiL L1 PB Ack
31         ...
32
33  |= AG(HiL & ~Ack -> AF horn).
34  The assertion is TRUE.
35
36  |= AG(HiT & ~Ack -> AF horn).
37  The assertion is TRUE.
38
39  |= AG (horn -> AX (horn | ~horn & PB)).
40  The assertion is TRUE.
41
42  |= AG(PB -> AF(Ack & ~horn)).
43  The assertion is TRUE.
44
45  |= EF(horn & EF(~horn & EF horn)).
46  The assertion is FALSE.
47
48  There is no counterexample for EF (horn & EF (~horn & EF horn))
49
50  |= AG(~horn -> EF horn).
51  The assertion is FALSE.
52
53  EF ~(~horn -> EF horn)
54         is true in state 1 because of the path:
55  State 1:
56  State 3:  horn HiT L2
57  State 8:  HiT L2 PB Ack
58
59  ~horn -> EF horn
60         is false in state 8 if:
61             1) ~~horn
62                    is false in state 8, AND
63             2) EF horn
64                    is false in state 8.
65
66  ~~horn
67         is false in state 8 because the following propositions are true:
68  ~horn
69
70  There is no counterexample for EF horn
71
72  |= AG(~HiL & ~HiT & ~PB -> AF ~Ack).
73  The assertion is FALSE.
74
75  EF ~(~HiL & ~HiT & ~PB -> AF ~Ack)
76         is true in state 1 because of the path:
77  State 1:
78  State 3:  horn HiT L2
79  State 8:  HiT L2 PB Ack
80  State 12:  HiT L2 Ack
81  State 9:  Ack
82
83  ~HiL & ~HiT & ~PB -> AF ~Ack
84         is false in state 9 if:
85         1) ~(~HiL & ~HiT & ~PB)
86                is false in state 9, AND
87         2) AF ~Ack
88                is false in state 9.
89
90  ~(~HiL & ~HiT & ~PB)
91         is false in state 9 because the following propositions are true:
92  ~HiL ~HiT ~PB
93
94  AF ~Ack
95         is false in state 9 because
96  EG ~~Ack
97         is true in state 9.
98  An example of such a path is:
99      State 11:  HiL L1 Ack
100     State 9:  Ack
101     ...
```

Figure 15. CTL model checker execution for the alarm acknowledge system.

initial condition of the circuit is that all variables are FALSE: that is, normally open contacts are open and normally closed contacts are closed. If the high-level sensor is activated, relay L1 actuates in rung 1 and the horn in rung 3 sounds. If the pushbutton is pressed, the Ack relay is closed in rung 4, and the horn is turned off in rung 3. Many possible states can be reached in this system. The possible paths associated with a process model are expressed by the state transition graph as shown in Figure 14. Two rules are used to convert the ladder diagram into the state transition graph: 1) variables that cause branching in the graph are the independent inputs in the ladder diagram, and the change of the independent variable (HiL, HiT or PB) between two states is shown on the arrow of the graph; 2) after each independent variable is changed, the ladder diagram and process models are used to update all the dependent variables. A simple *process model* of the operator's behavior for pressing the acknowledge pushbutton is "press the button once only after the horn is turned on." With those rules and the process model, the state transition graph is made and then converted into a Lisp-like input file as a system description for the CTL model checker.

One of the many possible desired *operating sequences* in the system is:

1) Once high level or high temperature is detected, then the horn is turned on.

2) Once the horn sounds, then the operator presses the acknowledge button.

3) Once the button is pressed, then the system is acknowledged and the horn goes off.

4) Repeat the above.

The system can be tested for this sequence by asking appropriate questions to the model checker. Figure 15 shows a partial trace of the model checker execution which tests the operability of the system.

The verification of the first operating sequence is described below. The CTL expression, shown in line 4 of Figure 15,

$$AG(\text{HiL} \rightarrow AF\,\text{horn})$$

checks all possible states in the system. It examines whether the horn sounds whenever the high level is detected. The result of this test as performed by the CTL model checker is FALSE as shown in lines 5 through 31. The counterexample shows that while the system is acknowledged, the horn is not turned on even if the high level is detected. This situation is normal. The above assertion is too strong. To exclude this case and to continue verifying the first operating sequence, the following assertion is used in line 33.

$$AG(\text{HiL} \ \& \ {\sim}\text{Ack} \rightarrow AF\,\text{horn})$$

This examines whether the horn sounds whenever the high level is detected (HiL) and the system is not acknowledged. The answer is TRUE as shown in line 34. This means that under the condition, the high-level detector and the horn behave correctly as a user required. Lines 36 and 37,

$$AG(\text{HiT} \ \& \ {\sim}\text{Ack} \rightarrow AF\,\text{horn})$$

show that if the temperature inside the storage tank is high under the condition, the horn sounds. By testing the above two assertions (lines 33-37), the first operating sequence is tested and the result is that the system behaves correctly as specified.

The second operating sequence is verified in line 39.

$$AG(\text{horn} \rightarrow AX(\text{horn} \,|\, {\sim}\text{horn} \ \& \ \text{PB}))$$

which means that after the horn sounds, either it stays on or it is turned off only if the pushbutton is pressed. The result, TRUE, means that the horn goes off only under the specified condition.

The third operating sequence, "once the operator presses the acknowledge button, the system is acknowledged and the horn goes off," is verified as shown in lines 42 and 43.

$$AG(\text{PB} \rightarrow AF(\text{Ack} \ \& \ {\sim}\text{horn}))$$

The result shows that the pushbutton always is able to return the system to the acknowledged state, and to silence the horn.

The next several assertions (lines 45-end) demonstrate examples of the system not following a user's requirement (the fourth operating sequence). The assertion in line 45,

$$EF(\text{horn} \ \& \ EF({\sim}\text{horn} \ \& \ EF\,\text{horn}))$$

tests whether the horn works for sequences of inputs. The result shows that once the horn is turned on and off, then the system does not recover to the initial state. This is clearly a problem for the integrity of this system because if a high-level

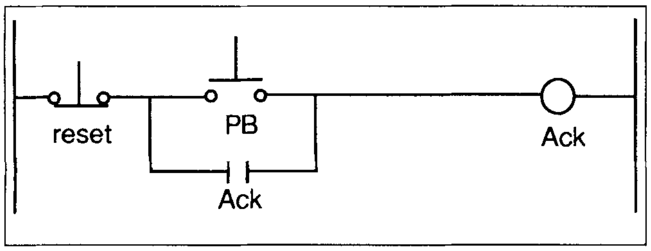

Figure 16. Revised fourth rung.

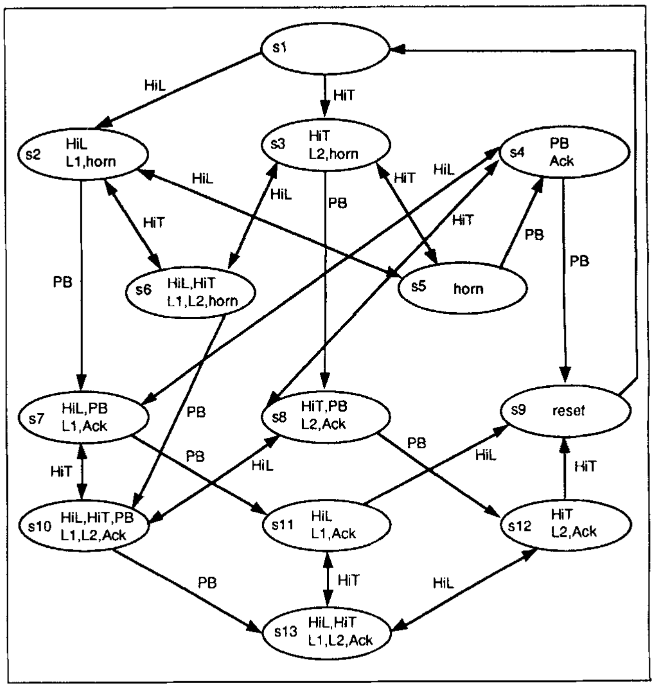

Figure 17. State transition graph for the revised alarm acknowledge system.

alarm comes in after the high-temperature alarm is acknowledged, the horn does not sound. Lines 50 through 101 are used to locate the cause of this failure. Line 50,

$$AG({\sim}\text{horn} \rightarrow EF\,\text{horn})$$

checks all states to detect if it is possible for the horn to sound later. The result of the test shows that there is no such path. As shown in lines 53 through 70, after the horn operates once (states 1, 3, 8) there is no path in which the horn is turned on again. The assertion in line 72,

$$AG({\sim}\text{HiL} \ \& \ {\sim}\text{HiT} \ \& \ {\sim}\text{PB} \rightarrow AF \ {\sim}\text{Ack})$$

asks if the high-level and the high-temperature sensors are reading normal and the pushbutton is not pressed, then eventually the acknowledge function should be returned to the initial condition (reset). The result shows that the system does not reset because of the infinite loop (states 1, 3, 8, 12, 9, 11, 9, ...).

For the horn to work properly for this situation, the ladder diagram should be revised. Figure 16 illustrates a possible solution by adding a normally closed reset button in the fourth rung of the ladder diagram. Figure 17 shows the corresponding state transition graph for the revised ladder diagram. A simple operator model for controlling the reset button is used:

$$\text{reset} = \text{Ack} \ \& \ {\sim}\text{HiL} \ \& \ {\sim}\text{HiT} \ \& \ {\sim}\text{PB}$$

which means that if the alarm has been acknowledged, all the alarm causing variables (HiL, HiT) are FALSE, and the acknowledge pushbutton is not pressed, then the operator should press the reset button; otherwise, do not press the reset button. This model is included in the new state transition graph.

```
 1        CTL MODEL CHECKER
 2  Taking input from alarm2.emc...
 3
 4  |= AG((HiL | HiT) & ~Ack -> AF horn).
 5  The assertion is TRUE.
 6
 7  |= AG (horn -> AX (horn | ~horn & PB)).
 8  The assertion is TRUE.
 9
10  |= AG(PB -> AF(Ack & ~horn)).
11  The assertion is TRUE.
12
13  |= EF(horn & EF(~horn & EF horn)).
14  The assertion is TRUE.
15
16  |= AG(~horn -> EF horn).
17  The assertion is TRUE.
18
19  |= AG(~HiL & ~HiT & ~PB -> AF ~Ack).
20  The assertion is TRUE.
```

Figure 18. CTL model checker execution for the revised alarm acknowledge system.

The same assertions are used to check the revised alarm system as shown in Figure 18. The two assertions in lines 33 and 36 of Figure 15 are combined and simplified as shown in line 4 of Figure 18. The execution of the CTL model checker with the same assertions in lines 4 to 20 indicates that the revised system does not have the reset error. This revised system has been tested for the other previous assertions and found to be satisfactory. This series of tests and revisions is one strategy for verifying and improving the integrity of sequential process control systems.

Conclusion

The integrity of chemical processing systems depends in part on the correctness of automatic control systems used in their operation. In the traditional approach to the verification of control systems, a series of manual tests is used to find errors in the system. Our work describes an automatic verification method that combines process models and the model checking method to identify errors in sequential chemical process control systems. This method consists of three components: *Process models* that describe systems; *assertions* that represent questions about the system; and a *model checker* that automatically determines whether the system operates as specified by assertions. The method has been used to verify an operating procedure in a combustion system and a ladder diagram in an alarm acknowledge system. These examples demonstrate that the method is able to express chemical engineering specifications and model the interactions between process equipment and the control software. The current method is limited to the verification of discrete event systems and depends on the development of process models. In addition, the generation of appropriate CTL assertions to assure system integrity depends on the user's interpretation of the system and has not been automated in this research.

Applications to industrially relevant problems will require:

- A more extensive library of state transition process models that include timers, delays, and counters.
- A more complete list of temporal logic assertions for the general testing of discrete chemical process control system safety and operability.
- A high-level language for stating assertions.
- A strategy for identifying the source of errors so that appropriate design changes could be proposed and evaluated.

Acknowledgment

Financial support from the National Science Foundation (Grants DMC-8616889 and CCR-8722633) is gratefully acknowledged.

Notation

A	=	all paths (CTL operator)
AP	=	the set of atomic proposition
E	=	there exists a path (or some paths) (CTL operator)
f	=	CTL formula
G	=	globally (CTL operator)
M	=	state transition structure
p	=	atomic proposition
$P(s)$	=	set of TRUE atomic propositions in state s
R	=	transition relation
S	=	set of states
s_i	=	state i
U	=	until (CTL operator)
X	=	next time (CTL operator)
~	=	NOT
&	=	AND
|	=	OR
$x \in S$	=	element x is a member of set S

Subscript

i	=	state number i

Literature Cited

Burch, J. R., E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang, "Symbolic Model Checking: 10^{20} states and beyond," *Proc. Symp. on Logic in Comput. Sci.* (June, 1990).

Chapman, K. G., and J. R. Harris, "Computer System Validation—Staying Current: Introduction," *Biopharm*, 30 (May, 1989).

Clarke, E. M., E. A. Emerson, and A. P. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications," *ACM Trans. on Programming Lang. and Sys.*, **8**(2), 244 (Apr., 1986).

Clarke, E. M., and O. Grumberg, "Research on Automatic Verification of Finite State Concurrent Systems," *Ann. Rev. Comput. Sci.*, **2**, 269 (1987).

Dhillon, B. S., and S. N. Rayapati, "Chemical-System Reliability: a Review," *IEEE Trans. on Reliability*, **37**(2), 199 (June, 1988).

Ho, Y. C., "Dynamics of Discrete Event Systems," *Proc. of the IEEE*, **77**(1), 3 (Jan., 1989).

IEEE Std 730-1984, "IEEE Standard for Software Quality Assurance Plans," IEEE, New York (1984).

IEEE Std 1008-1987, "IEEE Standard for Software Unit Testing," IEEE, New York (1987).

Laduzinsky, A. J., "PLCs Develop New Hardware and Software Personalities," *Control Eng.*, 53 (Feb., 1990).

Shaw, J. A., "Design your DCS to reduce Operator Error," *Chem. Eng. Prog.*, **87**(2), 61 (Feb., 1991).

Wallace, D. R., and R. U. Fujii, "Software Verification and Validation: an Overview," *IEEE Software*, **6**(3), 10 (1989).

Yamalidou, E. C., E. P. Patsidou, and J. C. Kantor, "Modeling Discrete-Event Dynamical Systems for Chemical Process Control," *Comput. & Chem. Eng.*, **14**(3), 281 (1990).

Manuscript received Dec. 4, 1990, and revision received Nov. 21, 1991.