1992

Symbolic model verifier for formal testing of discrete process control systems

Il Moon
Carnegie Mellon University

Gary J. Powers

Carnegie Mellon University.Engineering Design Research Center.

SYMBOLIC MODEL VERIFIER FOR FORMAL TESTING OF DISCRETE PROCESS CONTROL SYTEMS

Il Moon, Gary J. Powers

EDRC 06-125-92

Symbolic Model Verifier for Formal Testing of Discrete Process Control Systems

Il Moon* and Gary J. Powers
Department of Chemical Engineering
Carnegie Mellon University

May 27,1992

*** Author to whom correspondence should be addressed**

Abstract

We have developed a symbolic verification method for testing discrete chemical process control systems including process equipment and control system software and hardware. The method automatically determines if the system behaves as specified by safety and operability requirements. The method consists of: 1) a *system model* describing the process and its software, 2) *assertions* expressing user-supplied questions about the system behavior and 3) a *model checker* testing if the system model satisfies the assertions and supplying a counterexample if an error exists. The assertions are expressed using temporal logic operators for reasoning about occurrence of events in time. This verification method symbolically inspects the elements of the model. Compared with our previous explicit state enumeration method (Moon, 1991), this symbolic method can handle larger systems and be more complete in its system description. The method has been tested on an alarm system to uncover discrete event sequencing errors.

Introduction

The integrity of chemical process systems depends in part on the reliability of automatic control systems used in their operation. Traditionally, the test determining if the system works correctly is done manually based on human experience. In an attempt to replace the practice of manual testing, we have developed an automatic verification method which tests safety and operability of discrete chemical process control systems (Moon, 1991). In this paper, the method is further extended to solve more complex problems using *symbolic* computation techniques.

Numerous methods (Dhillon, 1988) have been used to test chemical processes, including hazard and operability studies (HAZOPs), failure mode and effect analysis (FMEA) and fault tree analysis (FTA). None of these methods have been widely used to test software imbedded in process control systems mainly because of the complexity of the control software. There are many techniques for testing software, including static analysis, program mutation and input space partitioning (Wallace, 1989), but these techniques have been used rarely in testing control processes mainly because of a lack of good models for the chemical processes.

We have developed an automatic verification method using the Extended Model Checker (EMC) to test safety and operability of both control software and chemical processing systems. EMC, originally developed by Clarke *et al.* (1986), automatically verifies finite state concurrent systems and has been applied to VLSI circuit testing. We have applied this method to chemical process control systems. The method uses a model-based approach which automatically determines if the discrete sequential system meets safety, reliability and operability requirements.

The method consists of a system description, assertions and a model checker. The system, including the process equipment and the control system hardware and software, is modeled using a state transition graph. Assertions are questions about the system behavior associated with safety and operability. These assertions are expressed using temporal logic operators, e.g. ~ (not), A (and), A (all), E (some), G (globally), F (in the future) for reasoning about the occurrence of events in time. The model checker tests if assertions are satisfied in the system and demonstrates a counterexample if an error exists. It can answer the following types of questions: "Is the valve vl

always open in the system, AG (v1) ?", "Is there a future situation in which fuel is flowing without flame, EF(fuel A -flame) ?", etc.

The complexity of control systems is a possible deterrent to use of EMC in large systems. Changing discrete input variables, e.g. temperatures, concentrations and tank levels, gives different behavior in the control system. For each combination we may simulate the behavior to test the safety and operability of the control system. However, the number of simulations required to test all combinations is too large even in a system of moderate size. The complexity grows as 2^n, where n is the number of binary inputs. One advantage of using EMC is that it is not necessary for the user to generate all possible situations since the method generates them automatically. The trouble with this method is that it can not handle large systems because EMC uses an explicit system description which includes all feasible states. The efficiency with respect to memory requirements has been improved to handle large systems by representing the system symbolically instead of explicitly and generating only necessary states for computation.

The Symbolic Model Checker (SMC) has been developed to improve the efficiency of EMC by Burch et al. (1990). SMC stores only the relationship between states by functions, so can handle larger systems. The EMC approach handles $10M0^6$ states (approximately 20 discrete variables). The SMC method has been used for VLSI tests with up to 10^{20} states (approximately 65 discrete variables). Advantages of using SMC are:

- Possibility of handling larger systems
- More direct modeling of the system

In this paper, we test the symbolic method on a chemical process control system, previously solved via EMC.

Model Checking Verification

An overview of the verification method using a model checker is shown in figure 1, where the system description and the assertions are inputs to the model checker. The *system description* is a model of the system to be tested. The model is derived from the process, operating procedures, control software, and the piping and instrumenting diagram (P&ID). The *assertions*

are questions about the system behavior associated with safety and operability coming from industrial standard checklists, process design specifications, or other methods like hazard and operability studies (HAZOP) or fault tree analyses (FTA). Assertions are expressed in computation tree logic (CTL) which is a prepositional, branching time temporal logic. The *model checker* searches the state space of the system model and determines the validity of assertions.

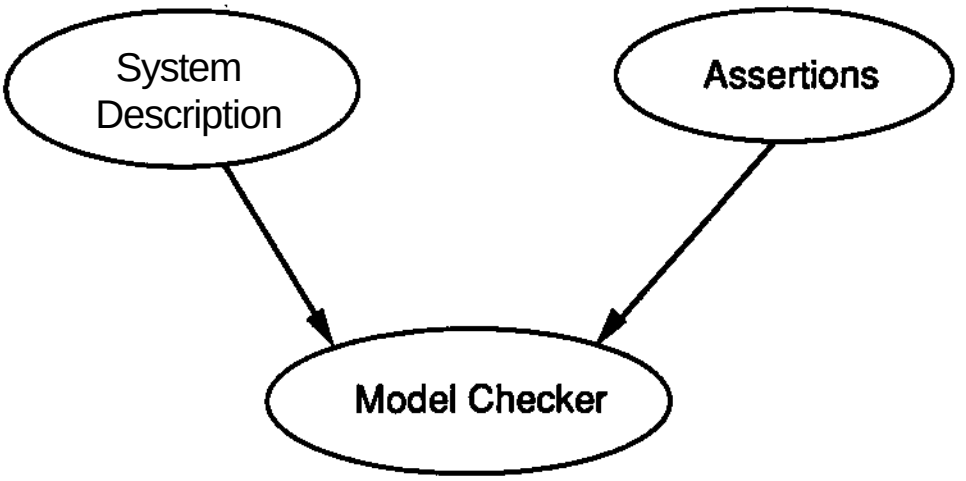


Figure 1. The overview of the CTL verification method

The next section describes the modeling of sequential systems and is followed by a description of CTL expressions and the symbolic model checking algorithm.

System Modeling

The system description includes the process equipment, human procedures and the control system hardware and software. The system description format used in EMC is a labeled state transition graph which explicitly shows all the system states. The format used in SMC is a implicit symbolic representation which describes only the relationship between states using boolean operators and time associated operators.

The two system description methods are illustrated by an example. Consider a sequential system including two discrete variables, al and a2, and the initial state is that both al and a2 are FALSE. The relationship between the variables in two states is that the value of al in the next state is the same as the value of a2 in the current state. This system is described in figure 2 as a state

transition graph where a circle indicates a state and an arrow denotes a state transition. Only the variables that have the value TRUE in a given state appear in the circle representing that state. The immediate successors of a state are the states that can be reached from the state in one step. For example, no value is shown in the state So of the graph because the values of both al and a2 are FALSE in the initial state. The successors of So are So and Si as shown in figure 2. According to the relationship, the value of al in successors of So is FALSE. The value of a2 in the states can be either TRUE or FALSE because a2 is not bounded by any relationship. These are shown in the successor states. The other states that follow this relationship are also given in figure 2. This description method as used in EMC explicitly represents all the system states.

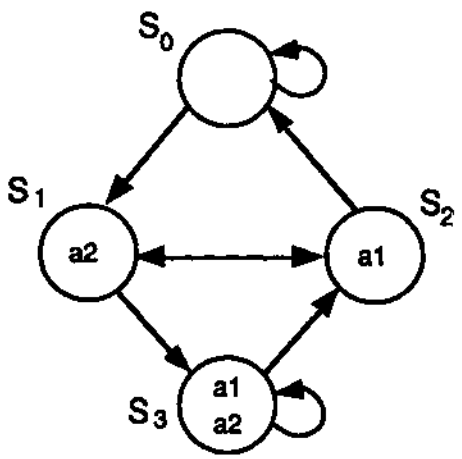


Figure 2. An example of a labeled state transition graph

An implicit symbolic system description used in SMC for the above example is:

Initial State
~a1 A ~a2
Transition Relationship
next(a1) = a2

where ~, A and next() mean NOT, AND and "next state" respectively. In the symbolic computation, only the initial condition and relationships between states are required for the system description. The symbolic method uses only the implicit relationships, and does not require all the system states explicitly.

Assertions

Assertions are questions about the system being verified. The assertions may be selected to test operability, safety and reliability features. These test assertions come from the system analyst, standard tests based on system specifications, or error types discovered in previous designs. Computation tree logic (CTL) formulae are used to express the assertions.

CTL combines both path quantifiers, A (for all computation paths) and E (for some computation path), and state quantifiers, G (always), F (sometimes), X (next time) and U (until). The formal syntax for CTL is:

(1) Every atomic proposition p e AP is a CTL formula.

(2) If fl and f2 are CTL formulae, then so are ~fl, H A G , AXf2, EXfl, A[f1 U f2] and E[fl U £2].

AP is the set of atomic propositions, where an atomic proposition is the state variable which denotes the property of interest and has either TRUE or FALSE value. The formula AXf 1 means that f1 holds in all immediate successors of the current state. EXfl means that f1 holds in some immediate successor. A[fl U f2] means that for every computation path 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. E[f 1 U f2] has the analogous meaning for some computation path.

When a temporal operator is prefixed by the *universal path quantifier* A, it indicates that the temporal property must hold over all possible computation paths beginning in the current state. The *existential path quantifier* E indicates that the condition expressed by the operator holds along some computation path beginning with the current state. Formulae involving the universal path quantifier can be expressed using the existential path quantifier and vice versa. For example, AXf is equivalent to -(EX ~f)« The following abbreviations are used in writing CTL formulae:

EF(f) ˢ E[TRUE U f] means that there is some path from so that leads to f; i.e. f holds *potentially.*

AF(f) ˢ A[TRUE U f] means that f holds in the future along every path from the initial state so; i.e. f is *inevitable.*

EG(f) ^{s} ~AF(~f) means that there is some paths from so on which f holds at every state,

AG(f) ^{s} ~EF(~f) means that f holds at every state on every path from so; i.e. f holds ***globally.***

Figure 3 shows how simple correctness properties can be represented using these operators, where the black circle and the white circle indicate that the atomic propositions of their states are TRUE and FALSE respectively. Figure 3 (a) shows an example that the CTL formula EFp is TRUE in the root state because a black circle exists in a successor state. More complex formulae 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. The expression EF EG (f) means that in the future there exists a path in which f is TRUE at each state along the path.

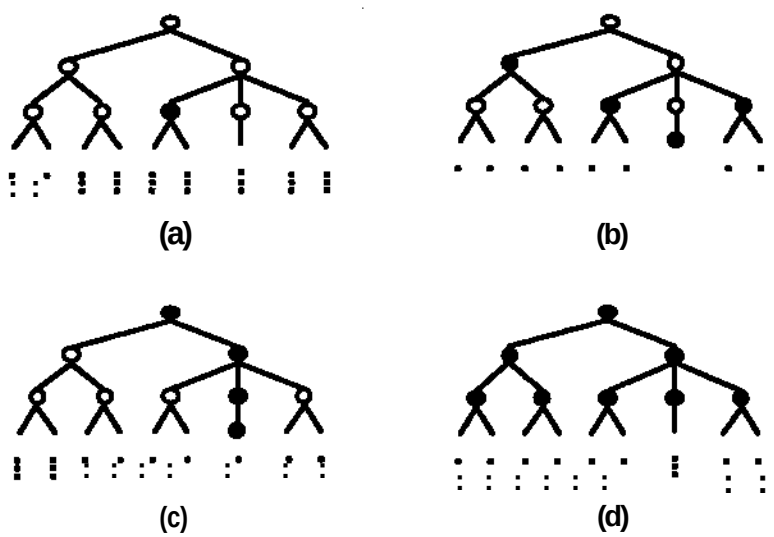


Figure 3. The CTL operators • = **p, o** = ^**p**
(a) EF(p): p potentially holds (b) AF(p): **p is inevitable**
(c) EG(p): p holds at every state in some path (d) AG(p): p is invariant

Examples of assertions expressed by the above CTL operators are AG (v1) and EF(fuel A aflame) which mean "Is valve vl always open in the system?" and "Is there a future situation in which fuel is flowing without flame?" respectively.

Binary Decision Diagram

Symbolic representation is necessary to describe the system model compactly and to use less computer memory. We use a Binary Decision Diagram (BDD) to describe the discrete system model symbolically. A **BDD** is a canonical form of boolean formulae (Bryant, 1986). The rooted, directed acyclic graph in figure 4 is an example of a **BDD** representing a boolean function / (xi,x2,x3) = (xi A X2) v X3, where A and v are logical AND and OR respectively. The following rule can be used to see / (1,0,1) = 1:1) trace a path from the root of the diagram to a leaf, and 2) at every node choose the branch directed by the value of the corresponding variable. These rules can be used to completely determine the function represented by a BDD. This representation is more compact than storing the whole truth table.

Bryant described details of algorithms for arranging boolean connectivities and composing functions. An algorithm for computing restrictions of functions is also given. The restriction of the function /(a, b, c, d) to a = 0 (written $/|_{a=o}$) is the 3-ary function g(b, c, d) = /(0, b, c, d). It is also possible to quantify over boolean variables. For example, the formula Ea /(a, b, c, d) is equal to $/(a, b, c, d)|_{a=o}$ v $/(a, b, c, d)|_{a=i}$.

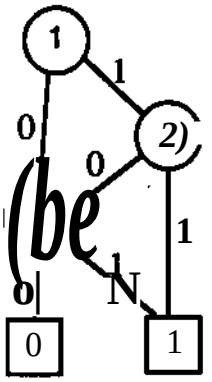


Figure 4. The Binary Decision Diagram, / = (xl A **X2)** V **X3**

Symbolic Model Checking

Model checking means determining whether a given formula / is satisfied in a state given a transition relation *R*. An algorithm has been developed that uses BDDs as its internal representation to avoid enumerating the elements of the model. The algorithm is defined by a function BDD which recurses over the structure of the formula. The function BDD takes two arguments: a formula / and a BDD. Their properties are that BDD(/, *R)* is true in a given state if and only if the formula / is true in that state.

The representation of the transition relation in a BDD is /?(V̄J, v̄y), where V̄J is the state before the transition, and v̄y is the state after the transition. The bar means vector variables. The state v̄y is a successor of v̄/ whenever the BDD is satisfied.

Assume that we have computed the BDD representing a subformula /, and wish to compute the BDD representing EX/. This formula is true in a state if and only if there exists a successor of that state which satisfies /. In other words, if the current state is v̄/, then there exists a truth assignment to the variables in v̄y which satisfies BDD(/, *R)* such that J?(v̄/f v̄y) is satisfied. Using boolean quantification, we can express this condition as:

$$BDD(EX/, R)(\bar{y}_i) = 3\bar{v}\bar{y}\ [tf(\bar{v}/, \bar{v}\bar{y})\ A\ BDD(/, R)(\bar{v}_j))].$$

In practice, we first relabel BDD(/, *R)* to use the variables of v̄y. Next the logical AND operation and the existential quantification operation are performed in the same pass over the BDDs. This is done to reduce the storage required for the intermediate results.

The formula EG/ states that there exists some path that / is globally true along beginning with the current state. This means that / is true in the current state and EG/ is true in some successor state. The algorithm for EG/ is given by

$$EG/ = /AEX(EG/).$$

The formula E[/ U g] means that there exists some path beginning in the current state in which g is true sometime in the future and / is true in all the proceeding states. This means that

either g is true in the current state or E[/ U g] is true in some successor state. The algorithm for E[/ U g] is given by

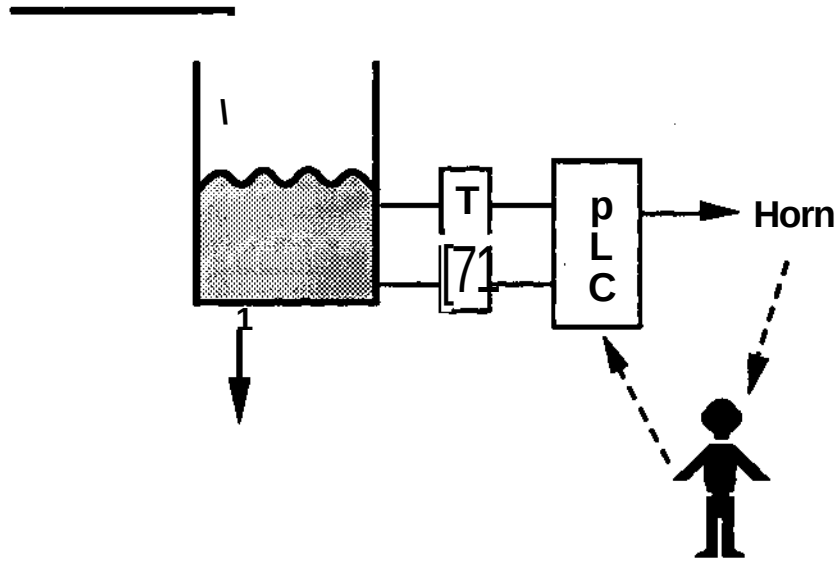
$$E[/Ug] = gv(/A\,EX(E[/\,U\,g])).$$

The logical operations (v, A, ~, ->) on BDDs can be used to compute BDDs which are true if and only if the formula is true. Since AX/, AG/ and A[/ U g] can all be rewritten using just the above operators, the above procedure covers the entire logic (Burch, 1990).

The model checker may also be used to provide a counterexample to a FALSE assertion. When the checker determines that an assertion is FALSE, it tries to find a path which demonstrates that the negation of the assertion is TRUE. This feature is quite useful for locating the source of an error. The following alarm system example illustrates the method and demonstrates its usefulness in the verification of discrete control systems at the detailed software and process model level.

An Alarm System Example

We use the same example as one of those in our previous paper (Moon, 1991). Consider an alarm system shown in figure 5, where the high level and the high temperature alarms of a storage tank are activated by a programmable logic controller (PLC), and the horn is acknowledged by an operator. A ladder diagram used for the PLC is shown in figure 6. 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 are placed on the rungs of the ladder. If the appropriate contacts are activated, a coil is energized and its associated relays are closed if they are normally open relays. For example, closing the high level contact (HiL) in rung 1 activates the relay coil LI in rung 1, and causes closing of the relay LI in rung 3.



FigureS. An alarm system

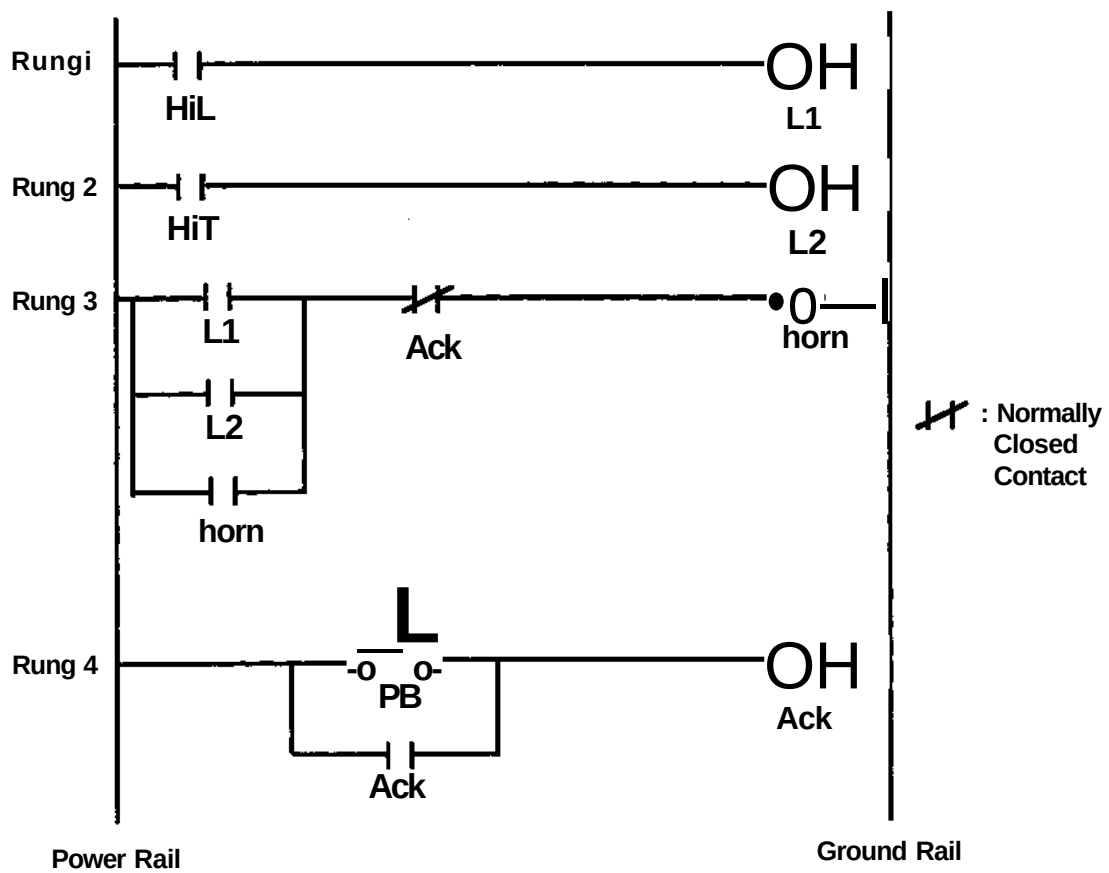


Figure 6. The ladder diagram for the alarm system

The ladder diagram in figure 6 includes two sensors (high level and high temperature), one pushbutton (PB) and one horn. The horn and **the** acknowledge relay Ack are latched in rung 3 and rung 4 respectively, i.e. 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** initial condition of the circuit is that all variables are FALSE, i.e. normally open contacts are open and normally closed contacts are closed. In the following two sections, this alarm system is verified using the explicit enumerating method and the symbolic method.

Verification using EMC

This section is a summary of the example solved using EMC (Moon, 1991). The system includes the flow diagram in figure 5, the ladder diagram in figure 6 and the process model of the operator's behavior. This given system is converted to a state transition graph as shown in figure 7.

The model checker tests the system by answering user's appropriate questions. Figure 8 shows a partial trace of the model checker execution which tests the operability of the system. The CTL expression, shown in line 33 of figure 8,

AG(HiL & -Ack -> AF horn)

checks all possible states in the system. This examines whether the horn sounds whenever 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(HiT & -Ack -> AF horn)

show that if the temperature inside the storage tank is high under the condition, the horn sounds. The testing results of the above two assertions (lines 33 -37) show that the system behaves correctly as specified. Line 39,

AG(horn -> AX(hom | -horn & PB))

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 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(PB -> AF(Ack & -horn))
```

The result shows that the pushbutton is always able to acknowledge the alarm and to silence the horn.

The next tests (lines 45-end) demonstrate examples of the system not following a user's requirement The assertion in line 45,

```
EF(hom & EF(~horn & EF horn))
```

tests whether the horn works for sequences of inputs. The result shows that once the horn is turned on and an operator presses the acknowledge button, then the system will not return to the initial state. This is clearly a problem for the integrity of this system because if a high level alarm comes in after the high temperature alarm is acknowledged, the horn will not sound. Lines 50 through 101 are used to locate the cause of this failure. Line 50,

```
AG(~hom -> EF horn)
```

is a subset of the previous assertion. This assertion checks for a possible path such that if the horn does not sound, then the horn can sound again. 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. Line 72,

```
AG(~HiL & ~HiT & ~PB -> AF ~Ack)
```

asks that "Once the high level and the high temperature are normal and the pushbutton is not pressed, then always in the future will the acknowledge function return to the initial condition (reset)?". The result shows that there is no way to reset this feature because of the infinite loop (states 1,3, 8,12,9,11,9,...).

In order for the horn to work properly for this situation, the ladder diagram should be revised. A possible solution is adding a normally closed reset button as shown in figure 9 to the ladder diagram in figure 6. A simple operator's model for controlling the reset button is used:

```
reset = ~HiL & -HIT & ~PB & Ack
```

which means that the operator should press the reset button when HiL and HiT are FALSE, the acknowledge pushbutton is not pressed and the alarm is acknowledged, and not press the reset button in other cases. This model is included in the new state transition graph. All the answers of the model checker execution for this revised system to the same assertions are TRUE. This means that the system works correctly as specified. This cycle of testing and revising is one possible formal method for verifying and improving the integrity of sequential process control systems.

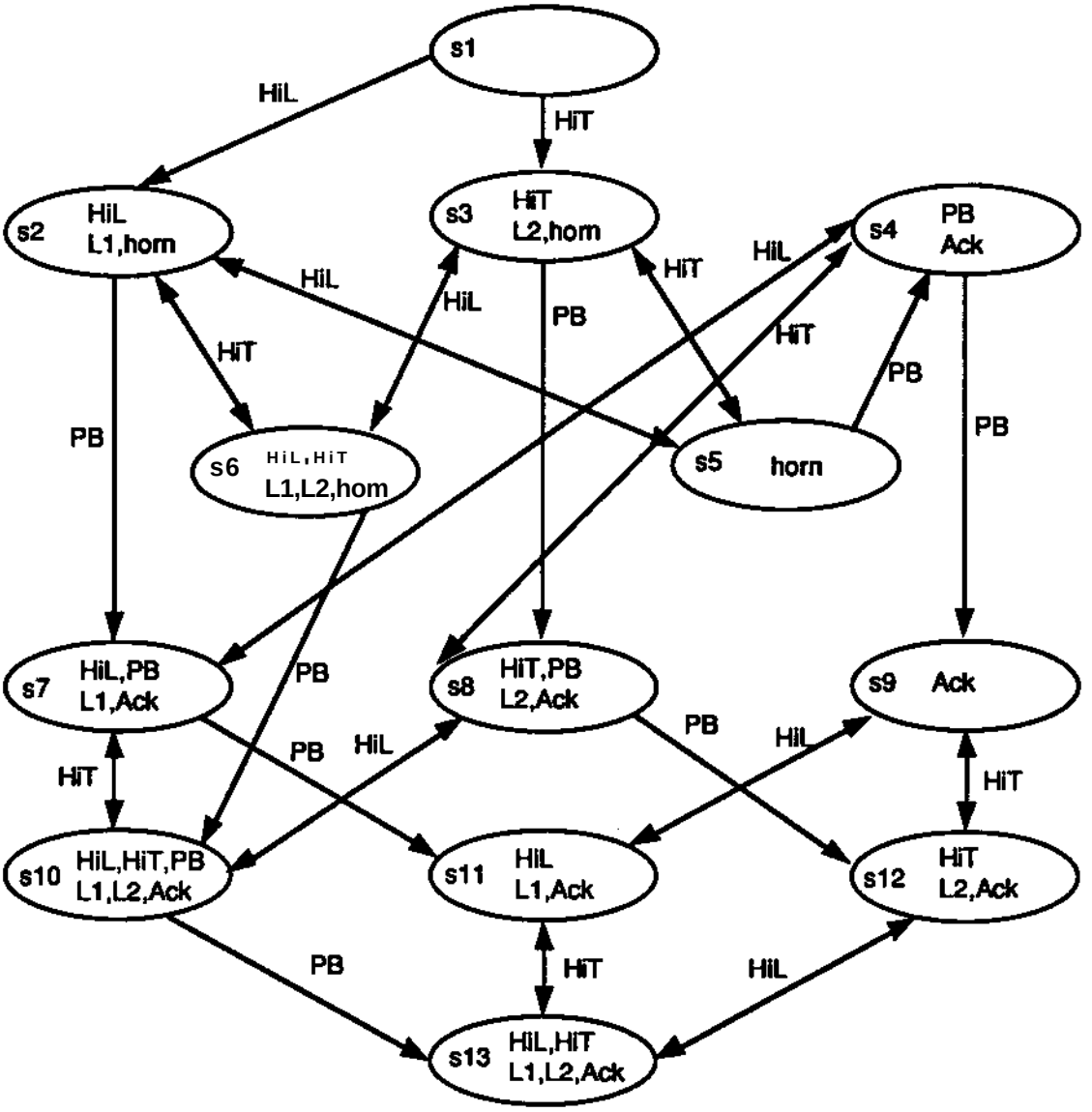


Figure 7. The state transition graph **for** the alarm system

```
1       CTL MODEL CHECKER
2  Taking input from aiarmi.emc...
...
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 I cani disprove 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 I cani disprove 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 8. The EMC execution for the alarm system

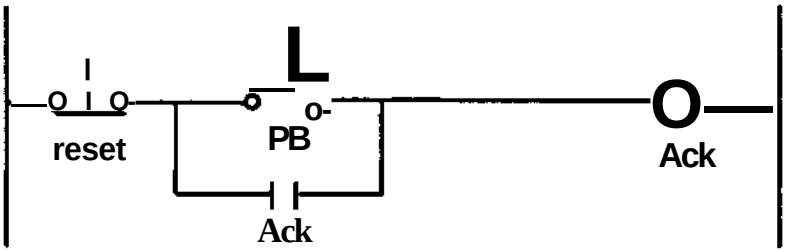


Figure 9. The revised fourth rung

Verification using SMC

This section describes the symbolic verification of the previous alarm system using the SMC method of Clarke et al. Figure 10 shows a part of the system description file for the first unrevised system given in figure 6. The numbers at the beginning of each line of figure 10 are added for reference and are not used by the program. All variables shown in the ladder diagram are declared in lines 2 and 3, and their initial condition is specified in lines 4 and 5.

Each transition relationship shown in line 7 to 10 corresponds to each rung in the ladder diagram in figure 6. All transition relationships are connected by AND because they should be satisfied simultaneously. Line 11 is a comment. Line 12 is a relationship that is not included in the ladder diagram but comes from a model of human operator's behavior. The model indicates that the pushbutton is pressed only after the horn is sounded and released only after the horn goes off. The same assertions as those used in the previous test are verified as shown in lines 13 to 19. The term "specification" in the figure has the same meaning as "assertion" in this case. The operator AG is automatically added by the model checker to the beginning of each assertion.

```
 1  MODULE main
 2  VAR
 3    HiL,HiT,PB,horn,Ack,L1 ,L2
 4  INIT
 5    -HiL & -HiT & -horn & -Ack & -PB & -L1 & -L2
 6  TRANS
 7       (next(L1) = HiL)
 8   & (next(L2) = HiT)
 9   & (next(horn) = (L1 | L2 | horn) & -Ack)
10   & (next(Ack) - PB | Ack)
11   - Process Model
12   & (next(PB) = horn)
13  SPECIFICATION
14   sped :- ((HiL | HiT) & -Ack) -> AF horn;
15   spec2 := PB -> AF(Ack & -horn);
16   spec3 := horn -> AX (horn | -horn & PB);
17   spec4 := EF(horn & EF(-horn & EF horn));
18   spec5 := -horn -> EF horn;
19   spec6 := -HiL & -HiT & -PB -> AF -Ack;
```

Figure 10. A system description input file for symbolic verification of an alarm system

Figure 11 shows a trace of the symbolic model checker execution which tests the same

assertions as those tested by the EMC execution. The output file includes only FALSE assertions. The result indicates that the assertions 1, 2 and 3 are TRUE, and the others are FALSE. The results of the tests are the same as the EMC execution, however the counterexamples arc different. The counterexamples show possible paths which demonstrate how the assertions are FALSE. The bottom of the counterexample list is the initial state and the top is the last state, where 0,1, and x mean FALSE, TRUE and "do not care" respectively.

For example, the assertion 5, ~horn -> EF horn, is FALSE in this system as shown in line 6 of figure 11, and one counterexample path is:

1. initial condition (line 13),
2. high temperature is detected (line 12),
3. relay L2 is on (line 11),
4. horn is on (line 10),
5. pushbutton is pressed (line 9),
6. relay Ack is on (line 8),
7. horn is off (line 7), and
8. after this sequence, there is no state that the horn turns on again (from the meaning of the assertion).

This is similar to the previous counterexample shown in lines 55 to 57 of figure 8. This result shows more steps than the previous test because intermediate states are compressed in the state transition graph.

The revised system is also tested using the symbolic method. The input file and the result are shown in figures 12 and 13 respectively. Line 13 of figure 12 is a model of an operator's behavior about pushing the reset button. The result of the symbolic model checker execution shown in figure 13 indicates that all the assertions are TRUE in this revised system. The result is the same as the explicit enumeration.

```
>smv alarmi.smv

FAIL SPECIFICATION <spec4>
    HiL,HiT,PB = 000     L1,L2,horn,Ack == 0000

FAIL SPECIFICATION <spec5>
    HiL,HiT,PB = XX1     L1,L2,horn,Ack =- XX01
    HiL,HiT,PB - XX1     L1,L2,horn,Ack =« XX11
    HiL.HiT.PB - XX1     L1,L2,horn,Ack >= XX10
    HiL,HiT,PB - XXO     L1,L2,horn,Ack .* XX10
    HiL.HiT.PB = XXO     L1,L2,horn,Ack >* 0100
    HiL,HiT,PB = 010     L1,L2,horn,Ack »« 0000
    HiL,HiT,PB = 000     L1,L2,horn,Ack == 0000

FAIL SPECIFICATION <spec6>
    HiL,HiT,PB - 000     L1,L2,horn.Ack == XX01
    HiL,HiT,PB = XX1     L1,L2,horn,Ack >' XX01
    HiL,HiT,PB - XX1     L1,L2,horn,Ack >. XX11
    HiL,HiT,PB = XX1     L1,L2,horn,Ack =* XX10
    HiL,HiT,PB = XXO     L1,L2,horn,Ack == XX10
    HiL,HiT,PB = XXO     L1,L2,horn,Ack *» 0100
    HiL.HiT.PB-010       L1,L2,horn,Ack => 0000
    HiL,HiT,PB = 000     L1,L2,horn,Ack == 0000
```

Figure 11. The SMC execution for the alarm system

```
MODULE main
VAR
  HiL,HiT,PB,horn,Ack,L1 ,L2,reset
INIT
  ~HiL & ~HiT & -horn & ~Ack & ~PB & ~L1 & -L2 & -reset
TRANS
    (next(L1) = HiL)
  & (next(L2) - HiT)
  & (next(horn) = (L1 | L21 horn) & -Ack)
  & (next(Ack) = PBI Ack)
- Process Model
  & (next(PB) - horn)
  & (next(reset) = -HiL&-HiT&~PB& Ack)
SPECIFICATION
 sped := ((HiL | HiT) & -Ack) -> AF horn;
 spec2 := PB -> AF(Ack & -horn);
 spec3 := horn -> AX (horn | -horn & PB);
 spec4 := EF(horn & EF(~horn & EF horn));
 spec5 := -horn -> EF horn;
 spec6 := -HiL & -HiT & -PB -> AF -Ack;
 spec7 := reset -> AF -Ack;
```

Figure 12. A system description for the revised alarm system

```
>smv alarm2.smv

SEARCH COMPLETE-> TRUE
```

Figure 13. The SMC execution for the revised alarm system

Conclusion

We have demonstrated an automatic symbolic verification method that uses Binary Decision Diagrams for testing the safety, reliability and operability of discrete chemical process control systems- Our previous verification method (Moon, 1991) has been improved in its system description capability and memory requirements for model checking. By comparison with VLSI experiments (Burch, 1990), this symbolic method will have the capability for handling large systems up to 10^{20} states. Currently, the symbolic verification method s limited to only discrete event systems, like the alarm system demonstrated in this paper. Another limitation is that appropriate CTL assertions depend on the user's interpretation of the system and has not been automated in this research. Larger examples are required to reveal the difference between the explicit method and the implicit symbolic method.

Notation

A= all paths (CTL operator)
AP = the set of atomic proposition
E = there exists a path (or some paths) (CTL operator)
f= CTLformula
G = globally (CTL operator)
p = atomic proposition
P = set of atomic proposition
R = transition relation
S = set of states
Si= state i
U = until (CTL operator)
X = next time (CTL operator)
~ = NOT
& = $_A$ = AND
l = v = OR
V = all
3 = there exist (or some)
x € S = element x is a member of set S
(x, y) e R = x is related to y by the relation of R
M, s 1= f = formula f is TRUE at state s in structure M

Subscripts

i= state number i

Literature Cited

Burch, J. R., E. M. Clarke, K. L. McMillan and D. L. Dill, "Sequential Circuit Verification Using Symbolic Model Checking", 27th ACM/IEEE Design Automation Conference, 1990.

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

Moon, I., G. J. Powers, J. R. Burch and E. M. Clarke, "Automatic Verification of Sequential Control Systems using Temporal Logic", accepted in *American Institute of Chemical Engineering Journal*, 1991.

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