APPLICATION OF FORMAL VERIFICATION AND FALSIFICATION TO LARGE-SCALE CHEMICAL PLANT AUTOMATION SYSTEMS

Blake C. Rawlings*[1,2], John M. Wassick[3], and B. Erik Ydstie[1]

[1]Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA
[2]Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, MI
[3]The Dow Chemical Company

Abstract

In this paper, we apply formal verification and falsification of temporal logic specifications to chemical plant automation systems. This can provide useful information about the behavior of the closed-loop hybrid system without requiring costly simulation or manual inspection. We extend previous work by applying a recently-developed approach to handle simultaneous invariance and reachability requirements, which require that the certain sets of states always be avoided, and that other sets of states always remain reachable. In addition, we develop a set of tests that can be generated automatically for a given control system, some of which can be tested using existing methods, and some of which combine invariance and reachability and therefore necessitate the new approach. In both cases, we work with abstractions of the automation logic in order to apply symbolic model checking to industrial-scale systems. We demonstrate the results using a series of small illustrative examples, and also report results from a case study using an industrial control system.

Keywords

Formal methods, Hybrid systems, Cyber-physical systems, Temporal logic, Plant automation, PLC, Supervisory control, Model checking

1 Introduction

1.1 Background and Motivation

Automating a modern chemical plant involves repeatedly making a large number of discrete decisions (once per sample interval) to guide the evolution of the process along the desired trajectory. This task is performed by a logical control system that observes and manipulates the process and continuous control system. The coupling between these three major components of a chemical plant results in a cyber-physical system, a type of hybrid (continuous and discrete) dynamical system, as described by Engell et al. (2000).

Due to the lack of systematic tools that can be used to design the discrete automation logic, the current industrial practice is to do so by hand, relying on a combination of engineer skill and intuition, semi-formal guidelines and best practices, rules of thumb, and simulation. Such an approach is both time consuming and error prone; this is a critical issue when it comes to the behavior of the plant, as described in the perspective by Leveson and Stephanopoulos (2013), which advocates the viewpoint that the various elements that make up the overall system are inextricably linked. The importance of developing tools to design hybrid process control systems is also highlighted in the perspective by Grossmann and Westerberg (2000). Analyzing existing systems is the first step toward systematically designing correct logic. The objectives of this work are to extend the existing methods for analyzing discrete chemical plant automation logic, and to apply the methods to industrial-scale systems.

1.2 Analysis of Logical Control Systems

A significant body of research has focused on verifying the correctness of logical control systems. This includes modeling programmable logic controllers (PLCs) so that a formal specification of the desired behavior can be verified (Moon 1994; Rausch and Krogh 1998; Canet et al. 2000; Gourcuff et al. 2008; Biallas et al. 2010; Darvas et al. 2013). The basic approach consists of the following steps:

1. Model the (discrete) dynamical behavior of the PLC as a (finite) state transition system.

2. Specify the desired behavior as a temporal logic formula.

3. Apply model checking to determine whether the model fulfills the specification.

This approach has previously been applied in the chemical processing industry (Moon et al. 1992; Probst et al. 1997; Bauer

*To whom all correspondence should be addressed: bcraw@umich.edu. This work was completed in part while the first author was a Ph.D. candidate at CMU, and in part while he was in his current position as a postdoctoral research fellow at U-M.

et al. 2004; Kim and Moon 2011). One of the main limitations of this approach is the *state-explosion problem* (Clarke and Grumberg 1987), which refers to the fact that the problem size increases rapidly as the number of discrete variables increases.

To overcome the state-explosion problem, Park and Barton (1997) proposed an *implicit Boolean state-space model* that describes the discrete logic. The implicit model is converted to an equivalent integer programming problem, which is checked for feasibility. The implicit model captures individual transitions, but not sequences of transitions from one reachable state to the next; as a result, it can be used to verify invariance properties (which require that the system never leave a particular fixed set of good states), but is conservative in the sense that it may fail to verify a correct system. For this reason, the standard approach based on symbolic model checking remains the most widely used method for analyzing logical control systems.

1.3 Contribution

The main contribution of this paper is to demonstrate the application of both verification and falsification to analyze the behavior of large-scale chemical plant automation systems. We do so in a way that accounts for the limitations (imposed by the hybrid dynamical nature of the systems) on the classes of specifications that can be verified or falsified. The verification methods are the same as those that have been applied previously; the difference is the particular specifications that we verify, and what the result indicates about the system. The falsification methods that we apply were developed recently, and allow a broader class of specifications to be addressed than in previous work. For both verification and falsification, we use a model-reduction technique to mitigate the state-explosion problem; the validity of the results is maintained, and the approach allows us to address industrial-scale systems. We also provide a set of specifications that can be used with the approach we describe to analyze a general automation system. We demonstrate our results through a series of illustrative examples, and report computational results from test cases provided by The Dow Chemical Company.

In Section 2, we introduce discrete automation systems and some specifications that they should satisfy. In Section 3, we show how to model standard control and automation logic. In Section 4, we describe the formal analysis methods that are applied to determine whether a system meets the specifications. In Section 5, we introduce a way to approximate large systems to avoid the state-explosion problem. In Section 6, we describe a pair of open source software tools that implement the methods presented in the paper. In Section 7, we apply the methods to an industrial case study. In Appendix A, we introduce background material that is used throughout the paper.

1.4 Notation

We use the notation ${a_i}^{(j)}$ throughout the paper to represent vectors. The subscript i indicates element i of the vector, and the superscript (j) is used to distinguish between different vectors (e.g., in a sequence or a set). In addition, we often partition a vector into other named variables, so that for the vector $a = \begin{pmatrix} b \\ c \end{pmatrix} \in \mathbb{R}^{n_a} \equiv \mathbb{R}^{n_b+n_c}$, $a_1 \equiv b_1$ and $a_{n_b+1} \equiv c_1$; we use the top-level variable (a in this case) and the inner variables (b and c in this case) interchangeably for convenience.

To simplify notation involving discrete variables, we use ‘‘0’’ and ‘‘1’’ to represent the integer values 0 and 1 as well as the Boolean values $false$ and $true$, respectively. That is, for $d \in \{0, 1\}^2$, we treat the expressions $d_1 + d_2 \geq 1$ and $d_1 \vee d_2$ as being equivalent.

The notation $F : X \rightrightarrows Y$ represents a set-valued map. This is equivalent to $F : X \to \mathcal{P}(Y)$, where $\mathcal{P}(Y)$ is the power set of Y.

2 Discrete Logic in Chemical Plants

2.1 Dynamics

We consider sample-and-hold control systems (SHCSs) produced by applying sample-and-hold control to continuous chemical processes, modeled in the hybrid dynamical system formalism from Section A.5 as:

$$
\begin{aligned}
x &= \begin{pmatrix} z \\ u \\ s \\ \tau \end{pmatrix} \in \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \times \{0,1\}^{n_s} \times [0, T] =: X \\
F(x) &= \begin{pmatrix} F_z(z, u) \\ \mathbf{0} \\ \mathbf{0} \\ 1 \end{pmatrix} \\
G(x) &= \begin{pmatrix} z \\ \left\{ \begin{pmatrix} u^+ \\ s^+ \end{pmatrix} \middle| \exists r \in \rho(z) : \begin{matrix} u^+ \in G_u(z, s^+) \\ s^+ = g_s(r, s) \end{matrix} \right\} \\ 0 \end{pmatrix} \\
D &= \{x \in X \mid \tau = T\} \\
C &= X \setminus D
\end{aligned}
\tag{1}
$$

where:

- z is a vector of continuous process state variables.
- u is a vector of continuous control variables.
- s is a vector of discrete control variables.
- τ is a timer variable that tracks the amount of time that has passed since the previous sample was taken.
- $F_z : \mathbb{R}^{n_z} \times \mathbb{R}^{n_u} \rightrightarrows \mathbb{R}^{n_z}$ represents the process dynamics.
- $\dot{u}$ and $\dot{s}$ are both $\mathbf{0}$ because the control variables only change value in discrete jumps when samples are taken.
- $z^+ = z$ because the process is continuous.
- $G_u : \mathbb{R}^{n_z} \times \{0,1\}^{n_s} \rightrightarrows \mathbb{R}^{n_u}$ is the continuous control law, which depends on the discrete control variables, s.

- $g_s : \{0,1\}^{n_r} \times \{0,1\}^{n_s} \to \{0,1\}^{n_s}$ is the discrete automation logic.
- $\rho : \mathbb{R}^{n_z} \rightrightarrows \{0,1\}^{n_r}$ returns discrete readings from the process and operators.
- $\dot{\tau} = 1$ and $\tau^+ = 0$ cause samples to occur every T time units.
- T is the sample time.

In such a system, the discrete control and automation logic is contained in the function g_s, which updates logical state of the control system in response to discrete readings. The discrete readings ρ come from the process (for example, by checking whether or not a continuous state variable is within a desired operating range) and the operators (in the form of discrete toggles on the operator's control interface).

The dynamics introduced by the discrete logic in a control system are fundamentally different than those introduced by the continuous control logic. A poorly-tuned PID controller (which would appear in G_u in Eq. (1)) will result in quantitatively degraded closed-loop performance, but the performance is often qualitatively similar to the performance that would be achieved using a well-tuned controller (i.e., the system is still stable, but converges to the set point more slowly). Minor changes in the discrete logic, however, usually produce qualitatively different behavior in the plant (i.e., a piece of equipment no longer activates under the correct conditions).

2.2 Process-Independent Tests

When checking the discrete automation logic in an SHCS, the specifications are in terms of the discrete control variables s. The atomic propositions from Section A.2 are then relational expressions involving those variables. Using these atomic propositions, temporal logic formulas can be constructed to describe certain properties of the desired system behavior.

Specifying the entire desired behavior of the closed-loop system is difficult for the same reasons that defining the control logic correctly is difficult. This leads to the goal of automatically generating specifications that describe part of the overall requirements that a control system must meet. We refer to these as process-independent tests (PITs), because they do not relate to the underlying chemical process (and can therefore be generated without knowledge of the process). Some PITs are listed in Table 1, and described in more detail in the remainder of this section.

A variable lock is the situation in which one of the discrete variables becomes stuck in either value, 0 or 1, without the possibility of ever changing. The requirement to avoid variable locks specifies that none of the output variables should ever become locked. This specification does not require that the variable ever changes value, only that the logic does not strictly prevent that from happening. For example, in the ideal case that a threshold alarm is never tripped, the corresponding discrete variable is always 0. This is not a variable lock unless there is no way the alarm would *ever* turn on (even in response to the threshold being violated).

Automation logic often involves explicitly defined operating modes, such as startup, react, and shutdown. This is described in detail by Park and Barton (2000). The system must always be in one (and only one) operating mode, which is specified by the requirement that the corresponding variables sum to 1. In addition to this, the control system should always be capable of reaching each of the operating modes, which is similar to the variable lock specification. As in the case of variable locks, the requirement that the operating modes remain reachable does not mean that any of them is actually reached, only that it is always possible to reach each of them. This specification is similar to the reachability requirement from Park and Barton (2000) (feasibility of a sequence).

The test for irrelevant logic is slightly different than the other specifications. The expected outcome is that removing part of the automation logic affects the behavior of the system in some way. This is done on a per-variable basis by introducing a new variable $s_i{}'$ with the same assignment logic as s_i, then removing part of the assignment logic for $s_i{}'$. The reachability specification $\mathsf{EF}(s_i \neq s_i{}')$ specifies that there should be some reachable state in which the original and modified variables have different values. If the specification is satisfied, then the logic that was removed is relevant (to the behavior of the system). If the specification is not satisfied, then the logic that was removed is irrelevant, and can be removed without modifying the system's behavior. It is common practice to intentionally include redundant (irrelevant) terms to clarify the logic, but sometimes this behavior is not intended.

3 Modeling PLC Programs

The methods we describe in this paper apply to any control system that has the form of Eq. (1). PLCs are often used in the chemical processing industry to implement the discrete logic of SHCSs. For this reason, we focus on PLC programs as the target of our analysis, and to give concrete examples; specifically, we use the Structured Text (ST) language defined in standard standard IEC 61131-3 (International Electrotechnical Commission 2013).

PLCs operate by repeating the following steps in a non-terminating loop:

1. Input scan: inputs to the PLC program (continuous and discrete values) are read from the plant.
2. Evaluate logic: the PLC logic is executed with the new inputs to update the outputs.
3. Output scan: the new outputs are applied to the plant.

In relation to the model in Eq. (1), step 1 corresponds to ρ, step 2 corresponds to G, and step 3 corresponds to the jump $x^+ \in G(x)$.

3.1 Translation to a Formal Model

We address PLC programs defined using a restricted subset of the ST language, similar to previous work (Rausch and Krogh 1998; Gourcuff et al. 2008). We assume the following restrictions:

Table 1. Process-independent tests.

Property to test	Specification
Avoid variable locks.	$\mathsf{AG}(\mathsf{EF}(s_i = 0) \wedge \mathsf{EF}(s_i = 1)) \quad i \in 1 \dots n_s$
All operating modes are reachable.	$\mathsf{AG}(\mathsf{EF}(s_j = 1)) \quad j \in J$
Operating modes are mutually exclusive.	$\mathsf{AG}(\sum_{j \in J} s_j = 1)$
Relevant logic.	$\mathsf{EF}(s_i \neq s_i') \quad i \in 1 \dots n_s$

- All assignments are to elementary Boolean or numeric variables.
- There are no loops (other than the PLC's loop over the entire program).
- There are no jumps (i.e., the program is a single routine).

That is, the ST program is a sequence of assignments, along with conditional branching.

Every variable that is assigned a value is an *output* of the program. Any variable that is not assigned a value anywhere in the program is an *input*. The output variables are the values that the control logic sets in order to influence the behavior of the plant. Variables that are not assigned values anywhere in the program are assumed to be readings from the plant, and therefore act as inputs to the PLC logic.

Example 1 (a simple PLC program)**.** Consider the ST program:

```
s1 := ABS(z1 - z2) > 0;
s2 := s3 OR r1;
s3 := s2 AND r2;
```

which produces the model:

$$
\begin{aligned}
x &= (z_1, z_2, s_1, s_2, s_3, \tau)^{\mathsf{T}} \in \mathbb{R}^2 \times \{0,1\}^3 \times [0, T] =: X \\
F(x) &= \begin{pmatrix} F_z(z) \\ \mathbf{0} \\ 1 \end{pmatrix} \\
G(x) &= \begin{pmatrix} z \\ \left\{ s^+ \,\middle|\, \exists r \in \rho(z) : \begin{array}{l} {s_1}^+ = r_3 \\ {s_2}^+ = s_3 \vee r_1 \\ {s_3}^+ = {s_2}^+ \wedge r_2 \end{array} \right\} \\ 0 \end{pmatrix} \\
\rho(z) &= \begin{pmatrix} \{0,1\} \\ \{0,1\} \\ |z_1 - z_2| > 0 \end{pmatrix} \\
D &= \{x \mid \tau = T\} \\
C &= X \setminus D
\end{aligned}
$$

The expressions $\rho_1 = \{0, 1\}$ and $\rho_2 = \{0, 1\}$ indicate the ρ_1 and ρ_2 are external inputs to the program, and might either be 0 or 1, regardless of the continuous state z. We have not explicitly defined the continuous dynamics F_z, and there are no continuous control variables u. Note that the implicit definition of g_s that arises when the assignment of one variable depends on a previous assignment in the program (as shown above for ${s_3}^+$, which depends on ${s_2}^+$) can always be converted to an explicit definition; this is described in more detail in Rausch and Krogh (1998) and Park and Barton (2000). In this example, the term ${s_3}^+ = {s_2}^+ \wedge r_2$ would be replaced with ${s_3}^+ = (s_3 \vee r_1) \wedge r_2$.

4 Formal Analysis

4.1 Abstraction as a Labeled Transition System

In order to analyze the automation logic of an SHCS $\mathcal{H}$ as in Eq. (1), we rely on the deterministic finite labeled transition system (LTS; refer to Section A.1):

$$(S, R, \Delta) \tag{2}$$

where:

$$
\begin{aligned}
S &= \{0,1\}^{n_s} \\
R &= \{0,1\}^{n_r} \\
\Delta &= \{(s, r, s^+) \mid s^+ = g_s(r, s)\}
\end{aligned}
$$

As described by Rawlings et al. (2015), the LTS in Eq. (2) is an abstraction of the (nondeterministic, infinite-state) system that represents the behavior of $\mathcal{H}$. The abstraction models the response of the discrete logic to any of the possible sequences of inputs. The result is that the abstraction overapproximates the behavior of the closed-loop system; in the actual system, whether or not a particular sequence of inputs can occur depends on the continuous dynamics.

4.2 Verification

Given an SHCS and its LTS abstraction, it is possible to directly verify certain classes of specifications by analyzing the LTS. One such class of specifications is ACTL, which is described in Section A.2. For a specification that is not contained in this class, such as a CTL specification that includes E, verification of the LTS abstraction does not imply verification of the SHCS. This includes even simple reachability requirements such as $\mathsf{EF}(p)$.

Of the PITs defined in Table 1, the requirement that the operating modes be mutually exclusive is an ACTL specification. Therefore, it is eligible for verification using the abstraction. In addition, the negation of a relevant logic specification, $\neg\mathsf{EF}(s_i \neq s_i') \equiv \mathsf{AG}(s_i = s_i')$, is an ACTL specification. Verifying $\mathsf{AG}(s_i = s_i')$ guarantees that the logic removed from s_i' is irrelevant. The other specifications all include both invariance and reachability, so they cannot directly be verified by applying model checking to the LTS abstraction.

Algorithm 1 is a simplified version of the standard abstraction-based approach for verifying ACTL specifications in hybrid systems, applied to SHCSs (Chutinan and Krogh

2001; Clarke et al. 2003). If the verification fails, we do not attempt to refine the abstraction, as in Chutinan and Krogh (2001) and Clarke et al. (2003), or interpret the counterexample, as in Probst et al. (1997). Refinement of the model requires a detailed model of the hybrid dynamics, which is often difficult to obtain and computationally costly to analyze for large industrial systems. Manual inspection of the counterexample amounts to informal abstraction refinement, which is difficult for the same reasons, and does not have the benefit of being algorithmically sound.

Algorithm 1: Verification of ACTL specifications.

Input : SHCS $\mathcal{H}$ and ACTL specification θ
Output : ''$\mathcal{H}$ satisfies θ'' or ''Unknown''
$(S, R, \Delta) \leftarrow$ LTS abstraction of $\mathcal{H}$ as in Eq. (2)
if (S, R, Δ) satisfies θ **then**
  return ''$\mathcal{H}$ satisfies θ''
else
  return ''Unknown''

Example 2 (ACTL specification verified)**.** Consider the operating sequence depicted in Figure 1, which shows the desired paths through a set of operating modes. This behavior is enforced by the following automation logic:

$$s = (s_1, s_{2a}, s_{2b}, s_{3a}, s_{3b}, s_4, s_a, s_b)^{\mathsf{T}}$$

$$r = \begin{pmatrix} r_1 \\ r_a \end{pmatrix}$$

$$\Delta = \left\{ (s, r, s^+) \;\middle|\; \begin{aligned} {s_1}^+ &= (s_1 \wedge \neg r_1) \vee (s_4 \wedge r_1) \\ {s_{2a}}^+ &= (s_{2a} \wedge \neg r_1) \\ &\quad \vee (s_1 \wedge r_1 \wedge {s_a}^+) \\ {s_{2b}}^+ &= (s_{2b} \wedge \neg r_1) \\ &\quad \vee (s_1 \wedge r_1 \wedge {s_b}^+) \\ {s_{3a}}^+ &= (s_{3a} \wedge \neg r_1) \\ &\quad \vee ((s_{2a} \vee s_{2b}) \wedge r_1 \wedge {s_a}^+) \\ {s_{3b}}^+ &= (s_{3b} \wedge \neg r_1) \\ &\quad \vee ((s_{2a} \vee s_{2b}) \wedge r_1 \wedge {s_b}^+) \\ {s_4}^+ &= (s_4 \wedge \neg r_1) \\ &\quad \vee ((s_{3a} \vee s_{3b}) \wedge r_1) \\ {s_a}^+ &= \neg (s_4 \wedge r_1) \\ &\quad \wedge ((s_1 \wedge r_1 \wedge r_a) \vee s_a) \\ {s_b}^+ &= \neg (s_4 \wedge r_1) \\ &\quad \wedge ((s_1 \wedge r_1 \wedge \neg r_a) \vee s_b) \end{aligned} \right\}$$

$$s^{(0)} = (1, 0, 0, 0, 0, 0, 0, 0, 0)^{\mathsf{T}}$$

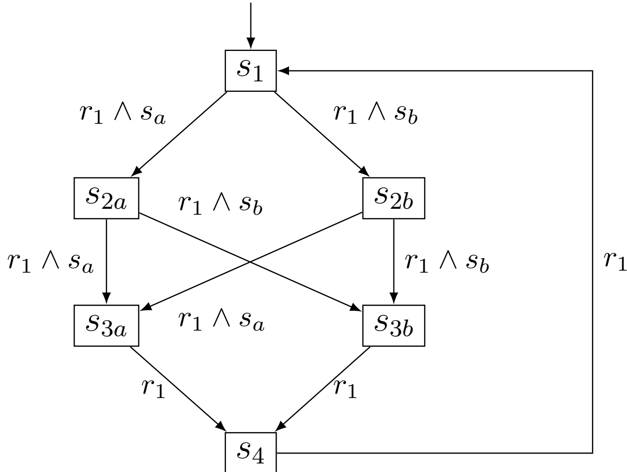

Figure 1. Sequence between operating modes.

In this simple example, the inputs do not relate to the continuous state of the process (r_1 and r_a are operator inputs to advance to the next operating mode and select ''recipe a'', respectively), so the discrete part of the automation system is decoupled from any continuous dynamics. The PIT for mutually exclusive operating modes in this example is the specification $\mathsf{AG}\left(\sum_J (s_j) = 1\right)$, where $J = \{1, 2a, 2b, 3a, 3b, 4\}$, which is verified by Algorithm 1.

Note that the states described by $((s_{2a} \vee s_{2b}) \wedge s_a \wedge s_b)$ are stable according to the definition in Park and Barton (1997), but lead to states in which $(s_{3a} \wedge s_{3b})$ holds, which violates mutual exclusivity. As a result, implicit model checking does not verify the specification, even though it is satisfied. The reason is that implicit model checking does not examine paths, starting in the initial state, consisting only of reachable states. The states in which $((s_{2a} \vee s_{2b}) \wedge s_a \wedge s_b)$ holds are not reachable, so the fact that they lead to bad states does not actually impact the specification. If s_b is replaced by $\neg s_a$ in the assignment logic for variables s_{2b} and s_{3b} in the control program, then the system's behavior is unchanged, but implicit model checking correctly verifies the specification. This highlights the conservative nature of implicit model checking, which is what allows for the reduction in computational effort required.

4.3 Falsification

Falsification of a class of specifications that combine invariance and reachability (CIR) in SHCSs is described in Rawlings (2016). These specifications have the form

$$\mathsf{AG}\left(\bigwedge_{i \in I} p_i \wedge \bigwedge_{j \in J} \mathsf{EF}\,(p_j)\right)$$

where the p_i and p_j do not contain any additional temporal operators (i.e., they are composed of atomic propositions and Boolean operators; refer to Section A.2). In addition to CIR specifications, any specification that is the negation of an ACTL specification can be falsified by verifying the ACTL specification. Consider the specification $\mathsf{EF}(p)$, which is equivalent to $\neg\mathsf{AG}(\neg p)$; falsifying $\mathsf{EF}(p)$ is equivalent to verifying $\mathsf{AG}(\neg p)$ using Algorithm 1.

The PITs in Table 1 that are CIR specifications are the requirement to avoid variable locks, and the requirement that all operating modes remain reachable. In addition, negating the relevant logic specification results in $\mathsf{AG}(s_i = s_i{}')$, which is also a CIR specification. Falsifying $\mathsf{AG}(s_i = s_i{}')$ amounts to verifying the original relevant logic specification. Each of these CIR specifications can be falsified by applying the results from Rawlings (2016).

Algorithm 2 is a simplified version of the method in Rawlings (2016) that does not include any reachability search in the hybrid system. In Algorithm 2, $SupervisorSynthesis$ is the standard supervisor synthesis algorithm from supervisory control (see Section A.4), which computes the maximally-permissive supervisor, given an LTS, a set of controllable events, and a specification; the expression $\Gamma/(S, R, \Delta)$ represents the closed-loop system in which the supervisor, Γ, is controlling (S, R, Δ) by disabling events. The continuous dynamics and any unmodeled discrete dynamics impact the sequences of inputs that can occur in the same way that a supervisor disables events in supervisory control (refer to Section A.4). An event r is disabled in state s of the LTS abstraction if there is no solution to the original SHCS that produces $r \in \rho(z)$ while the discrete part of the state is equal to s. To account for this, we treat each of the input readings $r \in R$ in the LTS abstraction as a controllable event, meaning that it is possible that r is prevented from occurring in a state s by some unmodeled behavior of the original SHCS. Thus, we set $R_c = R$ when analyzing the LTS in order to explore not only all possible sequences of inputs that might occur, but also all possible restrictions thereupon that the hybrid dynamics might impose. As with Algorithm 1, we omit the further analysis that involves the continuous dynamics, which is included in Rawlings (2016).

Algorithm 2: Falsification of CIR specifications (simplified).

Input : SHCS $\mathcal{H}$ and CIR specification θ
Output : "$\mathcal{H}$ does not satisfy θ" or "Unknown"
$(S, R, \Delta) \leftarrow$ LTS abstraction of $\mathcal{H}$ as in Eq. (2)
$R_c \leftarrow R$
$\Gamma \leftarrow SupervisorSynthesis((S, R, \Delta), R_c, \theta)$
if $\Gamma/(S, R, \Delta)$ does not satisfy θ **then**
  return "$\mathcal{H}$ does not satisfy θ"
else
  return "Unknown"

Example 3 (CIR specification falsified)**.** Consider a batch reactor with the sequence of four operating modes shown in Figure 2. and the reaction $A \rightarrow B$. The corresponding SHCS produces the LTS abstraction:

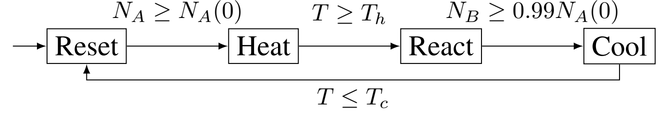

Figure 2. Operating mode sequence for a batch reaction.

$$s = (s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8)^{\mathsf{T}}$$

$$r = (r_1, r_2, r_3, r_4, r_5)^{\mathsf{T}}$$

$$\Delta = \left\{ (s, r, s^+) \;\middle|\; \begin{aligned} s_1{}^+ &= s_5 \wedge r_1 \\ s_2{}^+ &= s_6 \wedge r_2 \\ s_3{}^+ &= s_7 \wedge r_3 \\ s_4{}^+ &= s_8 \wedge r_4 \\ s_5{}^+ &= (s_5 \wedge \neg(r_5 \wedge s_1{}^+)) \\ &\quad \vee (s_6 \wedge s_2{}^+ \wedge r_5) \\ s_6{}^+ &= (s_2 \wedge \neg(r_5 \wedge s_2{}^+)) \\ &\quad \vee (s_7 \wedge s_3{}^+ \wedge r_5) \\ s_7{}^+ &= (s_3 \wedge \neg(r_5 \wedge s_3{}^+)) \\ &\quad \vee (s_8 \wedge s_4{}^+ \wedge r_5) \\ s_8{}^+ &= (s_4 \wedge \neg(r_5 \wedge s_4{}^+)) \\ &\quad \vee (s_5 \wedge s_1{}^+ \wedge r_5 \wedge s_2{}^+) \end{aligned} \right\}$$

$$s^{(0)} = (0, 0, 0, 0, 0, 0, 0, 1)^{\mathsf{T}}$$

with the relationship between the LTS variables and Figure 2 shown in Table 2.

Table 2. Description of the variables in Example 3.

Variable	Meaning
s_1	reactor is cool
s_2	reaction is complete
s_3	reactor is hot
s_4	reactor can be reset
s_5	"Cool" mode is active
s_6	"React" mode is active
s_7	"Heat" mode is active
s_8	"Reset" mode is active
r_1	$T \leq T_c$
r_2	$N_B \geq 0.99 N_A(0)$
r_3	$T \geq T_h$
r_4	$N_A \geq N_A(0)$
r_5	operator selected the next mode

The specification $\mathsf{AG}(\mathsf{EF}(s_5) \wedge \mathsf{EF}(s_6) \wedge \mathsf{EF}(s_7) \wedge \mathsf{EF}(s_8))$ comes from the PIT that each of the operating modes should always be reachable. The specification is a CIR specification, so Algorithm 2 can be applied to check whether the SHCS violates it. Applying Algorithm 2 indicates that the specification is violated. This is explained by the condition to move from $s_5 = 1$ ("Cool") to $s_8 = 1$ ("Reset"), which requires that $s_2 = 1$ ("Reacted"). The variable s_2 is only set in the "React" mode, so it is always 0 in the "Cool" mode; this prevent the system from returning to the "Reset"

mode after completing the reaction. Note that all the operating modes are reachable from the initial state, $s^{(0)}$, and the problem is that they do not always *remain* reachable. Thus, detecting this behavior requires analyzing a CIR specification that combines invariance and reachability. Finally, the result from Algorithm 2 guarantees that the SHCS does not meet the specification, regardless of the continuous dynamics.

5 Mitigating the State-Explosion Problem

To gain the performance benefit of cone-of-influence (COI) reduction (described in Section A.3) when analyzing systems in which all of the model variables appear in the COI, abstraction can be applied. Removing a variable's transition logic and replacing it with a free input to the model results in an abstraction. More importantly, breaking the connection to the variables that appeared in the transition logic potentially allows for COI reduction. Because the reduced model is an abstraction, the same theoretical results apply as discussed in Section 4: ACTL specifications can be verified, and CIR specifications can be falsified.

Example 4 (CIR falsified after simplification)**.** Consider the system from Example 3, and the LTS:

$$s = (s_1, s_2, s_5, s_6, s_7, s_8)^\mathsf{T}$$

$$r = (r_1, r_2, r_5, r_6, r_7)^\mathsf{T}$$

$$\Delta = \left\{ (s, r, s^+) \left| \begin{array}{rcl} {s_1}^+ &=& s_5 \wedge r_1 \\ {s_2}^+ &=& s_6 \wedge r_2 \\ {s_5}^+ &=& (s_5 \wedge \neg(r_5 \wedge {s_1}^+)) \\ && \vee(s_6 \wedge {s_2}^+ \wedge r_5) \\ {s_6}^+ &=& (s_2 \wedge \neg(r_5 \wedge {s_2}^+)) \\ && \vee(s_7 \wedge r_6 \wedge r_5) \\ {s_7}^+ &=& (r_6 \wedge \neg(r_5 \wedge r_6)) \\ && \vee(s_8 \wedge r_7 \wedge r_5) \\ {s_8}^+ &=& (r_7 \wedge \neg(r_5 \wedge r_7)) \\ && \vee(s_5 \wedge {s_1}^+ \wedge r_5 \wedge {s_2}^+) \end{array} \right. \right\}$$

$$s^{(0)} = (0, 0, 0, 0, 0, 1)^\mathsf{T}$$

in which state variables s_3 and s_4 from Example 3 have been replaced by inputs r_6 and r_7, respectively, and the inputs that only impacted s_3 and s_4 have been removed. This further abstraction of the model reduces the computational effort required for analysis, as the original system has 8 state variables and 5 input variables, while the reduced system has 6 state variables and 5 input variables. More importantly, the same result (that the specification $\mathsf{AG}(\mathsf{EF}(s_5) \wedge \mathsf{EF}(s_6) \wedge \mathsf{EF}(s_7) \wedge \mathsf{EF}(s_8))$ is violated) is still proven by applying Algorithm 2 to the reduced model.

6 Implementation

The methods that are presented in this paper have been implemented in a pair of software tools, st2smv[1] and SynthSMV[2], both of which are open source and freely available for academic and commercial use.

The first tool, st2smv, converts PLC programs written in Structured Text to formal models, as described in Section 3. It also includes options to generate some of the PITs from Section 2.2, handle certain function blocks such as delay timers, and compute abstractions that enable COI reduction as in Section 5.

The second tool, SynthSMV, analyzes the models produced by st2smv. SynthSMV is a modified version of the model checking solver NuSMV (version 2.6.0). The modifications made in SynthSMV implement supervisor synthesis on top of the efficient symbolic algorithms used in NuSMV for model checking. This allows for falsification of CIR specifications as in Section 4.3 in addition to verification as in Section 4.2, which is done using the model checking algorithms already present in NuSMV.

7 Case Study

We now apply our approach to two control systems which were provided by The Dow Chemical Company. Table 3 provides some basic details related to the size and complexity of the two systems. The first system, ''Unit A'', is a batch wash tank, and the second, ''Unit B'', is a batch reactor. The large number of discrete variables, in comparison to the relatively simple continuous control design (exhibited by the small number of PID loops), demonstrates the complexity of the discrete logic in a typical industrial control and automation system.

Table 3. Overview of the case study problem size. The columns list the number of PID loops, the number of discrete variables, the number of variables that represent operating modes, and the average size of the cone of influence of the variables.

Name	PIDs	Variables	Modes	COI
Unit A	5	236	22	570
Unit B	8	752	53	2462

To address the large problem size (for both Unit A and Unit B), we apply abstraction as in Section 5. There are many approaches for computing efficient abstractions of both discrete and hybrid systems, which are beyond the scope of this work. We apply the following abstraction procedure (which is implemented in st2smv) for each specification, to limit the size of the COI rooted at the set of variables that appear in the specification:

1. Set a limit on the number of state variables to include in the model.
2. Starting at 0, increment the COI search depth until the number of variables included in the COI exceeds the limit.
3. Replace each state variable that entered the COI at the previous search level with an input variable.

[1] https://pypi.python.org/pypi/st2smv

[2] https://bitbucket.org/blakecraw/synthsmv

The variable lock and relevant logic PITs were applied for each of the discrete control variables in each system for various COI size targets. The performance is summarized in Table 4. The state-explosion problem appears as the rapid increase in solution time as the COI size is allowed to increase. For each of the two systems, the largest target COI size listed represents roughly the largest value for which the methods in this paper could be applied. Comparing this limit (150) to the average original COI size in the two systems in Table 3, it is clear that COI reduction, made possible by analyzing abstract models, is critically important when it comes to analyzing the closed-loop behavior of industrial-scale systems.

Table 4. Abstraction and runtime information when applying the PITs to the case study. Each column lists the average value computed over the system variables. The columns list the target COI size when computing simplifying abstractions, the number of variables that were abstracted away, the resulting COI size after abstraction, and the time taken to analyze the PITs for each variable.

Name	COI Target	Abstr. var.	COI	Time/var. (s)
Unit A	25	8.7	16.2	5.9
Unit A	50	16.4	37.0	7.8
Unit A	100	22.3	83.5	17.2
Unit A	150	27.1	125.9	43.8
Unit B	25	16.3	27.1	16.4
Unit B	50	16.4	28.7	15.7
Unit B	100	36.2	69.4	21.3
Unit B	150	42.2	111.5	56.0

The results of applying the PITs to the sample systems are shown in Table 5. As expected, increasing the allowed COI size produces more conclusive results, at the expense of the increased execution time shown in Table 4. In both systems, each type of specification successfully detects the corresponding behavior, which supports the claim that the PITs described in Section 2.2 are relevant to industrial chemical plant automation systems. The trend of being able to prove more properties of the system behavior as the COI limit increases is expected to continue past the current COI limit of 150.

Table 5. PIT results from the case study. The columns list the number of results detected of each type.

Name	COI Target	Lock	Irrelevant	Relevant
Unit A	25	5	2	0
Unit A	50	8	12	0
Unit A	100	15	41	1
Unit A	150	15	44	2
Unit B	25	10	6	1
Unit B	50	10	8	1
Unit B	100	10	31	3
Unit B	150	10	50	4

The 5 variable locks detected in Unit A with the COI target set to 25 and the 10 in Unit B were the result of variables with assignment logic consisting of a single Boolean literal (`TRUE` or `FALSE`); this behavior was intended. The additional variable locks detected in Unit A for larger COI values and the instances of irrelevant and relevant logic, on the other hand, were not caused by such trivial dynamics. The detected behavior did not represent errors in the system, but this does indicate that detecting certain properties of the system's behavior requires a more detailed model of the system dynamics. Due to the proprietary nature of the control code, we have omitted the actual control logic.

8 Summary

In this paper, we have demonstrated the application of formal methods to analyze chemical plant automation systems. These systems are characterized by complex discrete logic which, coupled with continuous process dynamics, creates a large-scale hybrid dynamical system. Such systems are currently beyond the reach of systematic design tools, so instead we settled for proving certain aspects of the closed-loop behavior. To achieve this, we relied on automatically-generated process-independent tests (PITs) that involve both invariance and reachability requirements to obtain a high-level summary of a given control system. These PITs yielded promising results when applied to a case study consisting of two industrial automation systems, each from a batch process.

To address the state-explosion problem, which is the main difficulty in analyzing large-scale discrete systems, we combined symbolic algorithms for both model checking and supervisor synthesis with abstractions that enable COI reduction. This technique allowed our approach to scale to systems that could not be addressed using symbolic methods alone, with the trade off being that only a subset of the system's properties will be provable using an abstract model. Being able to easily scale a theoretically sound method to arbitrarily large systems is critically important, as neither failing for systems over a given size nor reporting false positives is desirable.

There are some promising directions in which this work can be extended. We applied a very simple technique to simplify the models and allow COI reduction; more sophisticated techniques could produce abstractions for which more of the system properties can be proven without increasing the size of the reduced model. The set of PITs we provided is by no means an exhaustive list of all the requirements that a plant automation system should meet; there remain opportunities both in developing more PITs, and in developing process-*dependent* tests that target particular aspects of a process's behavior.

9 Acknowledgements

Funding for this work was provided by The Dow Chemical Company and Industrial Learning Systems, Inc.

A Background Material on Discrete Event and Hybrid Systems

A.1 Discrete Event Systems (DES)

A DES is a system with a discrete state space that evolves by making discrete transitions in response to a sequence of events (Cassandras and Lafortune 2008). A DES can be

modeled by the labeled transition system (LTS), (S, R, Δ), where S is the state space, R is the set of event labels, and $\Delta \subseteq S \times R \times S$ is the set of transitions. When a DES is in state s and an event labeled r occurs, it makes a transition to a new state, s^+, such that $(s, r, s^+) \in \Delta$.

A.2 *Computation Tree Logic (CTL)*

The branching-time temporal logic CTL was introduced by Clarke and Emerson (1982). Two of the properties that can be described in CTL are *invariance* and *reachability*. The specification $\mathsf{AG}(p)$ is an invariance specification, which requires that the system never leave the set of states in which p holds. The specification $\mathsf{EF}(p)$ is a reachability specification, which requires that the system be able to reach the set of states in which p holds. Invariance and reachability are logical duals, i.e., $\mathsf{AG}(p) \iff \neg\mathsf{EF}(\neg p)$. In addition to temporal properties such as AG and EF, *atomic propositions* describe fundamental (atomic) properties of a system's state, and the Boolean operators $\wedge$, $\vee$, and $\neg$ can be used to combine and modify CTL formulas.

The *universal fragment* of CTL, called ACTL, is obtained by excluding the existential path quantifier, E. This assumes that the formulas are in *positive normal form*, meaning that the temporal operators are not directly negated (i.e., $\neg\mathsf{AG}(p)$ is first converted to $\mathsf{EF}(\neg p)$, which is clearly not an ACTL formula). ACTL is interesting primarily because if an ACTL formula is verified in an abstraction of a system, then it is guaranteed to hold in the actual system also (Clarke et al. 1994).

A.3 *Model Checking*

The objective of model checking is to determine whether a given model meets a temporal logic specification (Clarke et al. 1999). For a CTL specification, this is achieved by first computing the set of all states that satisfy the specification, then checking whether the initial state of the system is included in that set. If the specification holds in the system's initial state, then the model itself satisfies the specification; if not, then the model violates the specification.

The state-explosion problem, mentioned in Section 1.2, is specifically that the number of states in a DES can grow exponentially with the number of interacting components. One of the most successful methods for overcoming this problem is the use of binary decision diagrams (BDDs) to represent the model *symbolically* (Bryant 1986; McMillan 1992). Another technique to avoid the state-explosion problem is to consider only the components of a system that influence a given specification. That is, the model is built using only the state variables that appear in the specification, and the variables that (directly or indirectly) influence the value assigned to those state variables. In systems that contain multiple disconnected groups of variables, this cone-of-influence (COI) reduction can drastically improve the performance compared to the naïve approach (Clarke et al. 1999). However, for systems in which all the variable influence each other, COI reduction has no effect.

A.4 *Supervisory Control*

While the goal in model checking is to determine whether a DES meets a requirement, the goal in supervisory control is to modify the behavior of a DES to ensure that it does meet the requirement (Ramadge and Wonham 1987). Informally, given a DES modeled by the LTS (S, R, Δ), a set of *controllable events* $R_c \subseteq R$, and a specification θ, the *supervisor synthesis problem* is to compute a strategy for disabling events in R_c such that the modified DES satisfies θ. Supervisory control suffers from the same state-explosion problem as does model checking, but benefits from the same BDD-based symbolic algorithms.

A.5 *Hybrid Dynamical Systems (HDS)*

Systems that combine continuous and discrete dynamics are called hybrid dynamical systems, or simply hybrid systems. The following model represents a general hybrid system (Goebel et al. 2012):

$$\begin{cases} x \in C & \dot{x} \in F(x) \\ x \in D & x^+ \in G(x) \end{cases} \tag{3}$$

where x is the *state*, F is the *flow map*, G is the *jump map*, C is the *flow set*, and D is the *jump set*. The state varies continuously (flows) subject to the differential inclusion $\dot{x} \in F(x)$ when $x \in C$, and changes value discretely (jumps) subject to $x^+ \in G(x)$ when $x \in D$. The class of systems that can be modeled in the form of Eq. (3) includes other common classes of hybrid systems, such as hybrid automata, sample-and-hold control systems, and the mixed logical dynamical (MLD) systems proposed by Bemporad and Morari (1999). The main difficulty in analyzing the behavior of HDSs is that the reachability problem is only decidable for a severely restricted class of systems (Henzinger et al. 1995).

References

Bauer, N., Engell, S., Huuck, R., Lohmann, S., Lukoschus, B., Remelhe, M., and Stursberg, O. (2004). ''Verification of PLC programs given as sequential function charts''. In: *Integration of Software Specification Techniques for Applications in Engineering*. Vol. 3147. Lecture Notes in Computer Science, pp. 517–540. DOI: 10.1007/978-3-540-27863-4_28.

Bemporad, A. and Morari, M. (1999). ''Control of systems integrating logic, dynamics, and constraints''. In: *Automatica* 35, pp. 407–427. DOI: 10.1016/s0005-1098(98)00178-2.

Biallas, S., Brauer, J., and Kowalewski, S. (2010). ''Counterexample-guided abstraction refinement for PLCs''. In: *5th International Conference on Systems Software Verification*. USENIX Association.

Bryant, R. E. (1986). ''Graph-Based Algorithms for Boolean Function Manipulation''. In: *IEEE Transactions on Computers* C-35.8, pp. 677–691. DOI: 10.1109/tc.1986.1676819.

Canet, G., Couffin, S., Lesage, J.-J., Petit, A., and Schnoebelen, P. (2000). ''Towards the automatic verification of PLC programs written in Instruction List''. In: *IEEE International Conference on Systems, Man and Cybernetics* 4, pp. 2449–2454. DOI: 10.1109/icsmc.2000.884359.

Cassandras, C. G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. Springer. DOI: 10.1007/978-0-387-68612-7.

Chutinan, A. and Krogh, B. H. (2001). ''Verification of infinite-state dynamic systems using approximate quotient transition systems''. In: *IEEE Transactions on Automatic Control* 46.9, pp. 1401–1410. DOI: 10.1109/9.948467.

Clarke, E. M. and Grumberg, O. (1987). ''Avoiding the state explosion problem in temporal logic model checking''. In: *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*. DOI: 10.1145/41840.41865.

Clarke, E. M. and Emerson, E. A. (1982). ''Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic''. In: *Logics of Programs*. Vol. 131. Lecture Notes in Computer Science, pp. 52–71.

Clarke, E. M., Grumberg, O., and Long, D. E. (1994). ''Model checking and abstraction''. In: *ACM transactions on Programming Languages and Systems (TOPLAS)* 16.5, pp. 1512–1542. DOI: 10.1145/186025.186051.

Clarke, E. M., Grumberg, O., and Peled, D. A. (1999). *Model Checking*. The MIT Press.

Clarke, E. M., Fehnker, A., Han, Z., Krogh, B. H., Ouaknine, J., Stursberg, O., and Theobald, M. (2003). ''Abstraction and counterexample-guided refinement in model checking of hybrid systems''. In: *International Journal of Foundations of Computer Science* 14.4, pp. 583–604. DOI: 10.1142/S012905410300190X.

Darvas, D., Blanco Viñuela, E., and Fernández Adiego, B. (2013). *Transforming PLC Programs into Formal Models for Verification Purposes*. Tech. rep. CERN. URL: http://cds.cern.ch/record/1629275/files/CERN-ACC-NOTE-2013-0040.pdf.

Engell, S., Kowalewski, S., Schulz, C., and Stursberg, O. (2000). ''Continuous-discrete interactions in chemical processing plants''. In: *Proceedings of the IEEE* 88.7, pp. 1050–1068. DOI: 10.1109/5.871308.

Goebel, R., Sanfelice, R. G., and Teel, A. R. (2012). *Hybrid Dynamical Systems: Modeling, Stability, and Robustness*. Princeton University Press.

Gourcuff, V., de Smet, O., and Faure, J.-M. (2008). ''Improving large-sized PLC programs verification using abstractions''. In: *17th IFAC World Congress*. DOI: 10.3182/20080706-5-KR-1001.00857.

Grossmann, I. E. and Westerberg, A. W. (2000). ''Research challenges in process systems engineering''. In: *AIChE Journal* 46.9, pp. 1700–1703. DOI: 10.1002/aic.690460902.

Henzinger, T. A., Kopke, P. W., Puri, A., and Varaiya, P. (1995). ''What's Decidable about Hybrid Automata?'' In: *Twenty-seventh Annual ACM Symposium on Theory of Computing*, pp. 373–382. DOI: 10.1145/225058.225162.

International Electrotechnical Commission (2013). ''Part 3: Programming languages''. In: *Programmable controllers*. IEC Standard 61131. URL: https://webstore.iec.ch/publication/4552.

Kim, J. and Moon, I. (2011). ''Model Checking for Automatic Verification of Control Logics in Chemical Processes''. In: *Industrial & Engineering Chemistry Research* 50.2, pp. 905–915. DOI: 10.1021/ie100007w.

Leveson, N. G. and Stephanopoulos, G. (2013). ''A system-theoretic, control-inspired view and approach to process safety''. In: *AIChE Journal* 60.1, pp. 2–14. DOI: 10.1002/aic.14278.

McMillan, K. L. (1992). ''Symbolic Model Checking''. PhD thesis. Carnegie Mellon University.

Moon, I. (1994). ''Modeling Programmable Logic Controllers for Logic Verification''. In: *IEEE Control Systems Magazine* 14.2, pp. 53–59. DOI: 10.1109/37.272781.

Moon, I., Powers, G. J., Burch, J. R., and Clarke, E. M. (1992). ''Automatic verification of sequential control systems using temporal logic''. In: *AIChE Journal* 38.1, pp. 67–75. DOI: 10.1002/aic.690380107.

Park, T. and Barton, P. I. (1997). ''Implicit model checking of logic-based control systems''. In: *AIChE Journal* 43.9, pp. 2246–2260. ISSN: 1547-5905. DOI: 10.1002/aic.690430911.

— (2000). ''Formal verification of sequence controllers''. In: *Computers & Chemical Engineering* 23.11, pp. 1783–1793. DOI: 10.1016/s0098-1354(99)00327-0.

Probst, S. T., Powers, G. J., Long, D. E., and Moon, I. (1997). ''Verification of a Logically Controlled, Solids Transport System Using Symbolic Model Checking''. In: *Computers & Chemical Engineering* 21.4, pp. 417–429. DOI: 10.1016/0098-1354(95)00265-0.

Ramadge, P. J. and Wonham, W. M. (1987). ''Supervisory Control of a Class of Discrete Event Processes''. In: *SIAM Journal on Control and Optimization* 25.1, pp. 206–230. DOI: 10.1137/0325013.

Rausch, M. and Krogh, B. H. (1998). ''Formal verification of PLC programs''. In: *American Control Conference*. Vol. 1, pp. 234–238. DOI: 10.1109/ACC.1998.694666.

Rawlings, B. C. (2016). ''Discrete Dynamics in Chemical Process Control and Automation''. PhD thesis. Carnegie Mellon University.

Rawlings, B. C., Wassick, J. M., and Ydstie, B. E. (2015). ''Error Detection for Chemical Plant Automation Logic Using Supervisory Control Theory''. In: *IEEE Conference on Control Applications (CCA)*, pp. 376–381. DOI: 10.1109/CCA.2015.7320658.