Verification of Programmable Logic Controllers for Critical Infrastructures

McKenzy Heavlin

Abstract

Modern, automated industrial plants are typically managed by computers that allow for precise control of physical processes, but require significant safety and failsafe measures to ensure overall system safety. These strict requirements make these systems ideal candidates for formal verification. This project explores how to model these systems as hybrid automaton in the C2E2 verification engine and the methods for verifying system safety. We present an iterative verification methodology that builds complexity into the model while maintaining confidence in the verification results. We present output traces from C2E2's simulation engine as a complement to the verification process to ensure the system behaves as expected.

1. Introduction

Programmable logic controllers (PLCs) are computers optimized to provide scheduling, control, and adjustments for automated processes in an industrial setting. Historically, these devices have been utilized to provide low-level control at individual processes; however, modern PLCs can easily implement more complex control logic. These devices are present in many critical industry and infrastructure applications such as the power grid, wastewater treatment plants, and chemical plants.

Given the number of PLCs that operate within these critical sectors, paired with the increasing number of internet-of-things devices, there is a growing concern that these devices could operate in an unsafe manner. Arbitrary operation of these devices poses a significant threat to society: an impacted PLC could shut off power in the middle of a snow storm or accidentally overdose a town's water supply.

Due to these concerns, there has been work regarding how we can formally verify properties of these systems to ensure safe operation. The goal of this project is to better understand how formal verification is applied to these systems, what properties are appropriate for verification, what properties are not guaranteed by verification, and how this process can be improved.

. Correspondence to: McKenzy Heavlin <heavlin2@illinois.edu>.

ECE 584 Class Project, Spring 2025. Instructor: Prof. Huan Zhang

1.1. System Background

The water treatment system is a control loop that takes in untreated water, conducts a few rounds of filtering and chemical treatments, and outputs clean water ready for use. There are numerous works related to attacks on a secure water treatment facility, but most reference (10) which serves as the base system for this work. The goal was to verify that the pumps controlling the holding tank water level are correct and do not operate improperly, but to create the system in such a way that allows for easy extensions if desired.

1.2. Code Representation

The code base representing the system is developed using Python and contains an asynchronous client, an asynchronous server, and a few helper/input files. The main system was a part of Dr. David Nicol's Trustworthiness in Critical Infrastructure course in Fall 2024 and implementation of the water tank PLC control was submitted as part of an assignment for the semester. The code can easily be modified to introduce additional complexity if required for the CS 584 project. A copy of the code is available at `https://github.com/McKenzyHeavlin/uiuc_sp25_584`.

2. Related Work

Formal verification of PLC programs, while a niche research area, is a well studied field. This section briefly discusses similar articles that motivated this work and others that will be used as relevant background. The primary work motivating this project is (9) in which the authors use the SPIN model checker to verify properties of a Siemens PLC in an industrial environment. Specifically, the authors were interested in proving properties regarding fault tolerant performance and asynchronous scheduling. While the work was interesting, it is extremely limited in its methodologies as the verified system is proprietary.

(6), (8), (7) attempt to perform verification on ladder logic programs (a standard way of representing PLC programs) to

ensure proper performance and security of PLCs. (6) aims to detect attackers attempting to subvert normal operations of the ladder logic; they test their detection mechanism on a simple tank filling system (similar to the one proposed here). (8) use timed automata to model the cyclical behavior of a ladder logic program to ensure race conditions ("...when two instructions change a variable 'simultaneously'") do not exist within the program. How ladder logic programs are modeled can also vary across literature; for example, (7) use NuSMV to model the system and verify properties using computational tree logic.

As PLCs are utilized frequently in industry, some work attempts to address concerns related to scalability and automation. (4) attempts to provide an automated process to verify complex properties by creating formal transition rules from PLC languages (structured text and sequential function chart) to an "intermediate model" that can be easily converted to a given specification language. Likewise, (1) uses an intermediate model to remove implementation language dependencies and creates a tool that can process PLC programs as is with no manual conversions.

(11) create a mechanism to automatically detect safety violations using time event sequences and their experience with physical industrial control system testbeds. The idea of real-time verification is attractive in this setting but can be limited due to device resources. (5) addresses this by introducing an embedded hypervisor alongside the PLC; enabling the hypervisor to monitor all incoming traffic to the PLC, model the impact on the PLC, and then transfer safe values to their final location.

(3) presents a framework focused on verifying safety-critical railway interlocking systems using temporal first-order logic and model-checking techniques to ensure the correctness of ladder logic programs used in Siemens Rail Automation systems. However, their work emphasizes the need for domain-specific adaptations of formal methods to make them applicable in real-world industrial settings. Finally, (12) introduces a system that advances formal verification to the binary level for PLC control logic in industrial IoT contexts. By employing reverse engineering to construct control flow graphs and leveraging model optimization techniques, the system efficiently addresses compile-time errors and real-time runtime logic checks to ensure security and reliability in PLC systems.

3. Methodology

This section presents the process of simplifying the water tank to an environment able to be modeled by a hybrid automaton, a discussion of the verification tool utilized, and the chosen approach for modeling the system and ensuring correctness of the methodology.

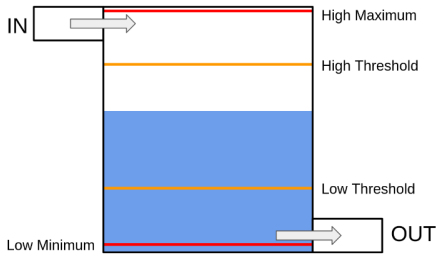


Figure 1. Tank simplification of a larger waste water treatment plant.

3.1. Problem Simplification & Automaton

Real-world waste water treatment plants can be a complex system with many different pumps, sensors, actuators, and filters; this makes them difficult to model using formal models. To address this challenge, this work begins from a simple water tank, shown in Figure 1, with the following characteristics:

- **IN**: The boolean state of water flowing into the system. When high, the valve is open and water flows into the tank; when low, the valve is closed and no water flows into the tank.
- **OUT**: The boolean state of water flowing out of the system.
- **High Maximum**: The maximum amount of water the tank can hold. Water levels above this value result in the tank overflowing.
- **High Threshold**: An operator set threshold to warn the control system that the water level is near the maximum.
- **Low Minimum**: The minimum amount of water the tank can hold.
- **Low Threshold**: An operator set threshold to warn the control system that the water level is near the minimum.

These characteristics are enough to begin the process of converting the system into a hybrid automaton for verification and enable the system to be extensible if desired. For example, future work could add a chemical pump that controls the chemical dose added to the water; this could then be used in the hybrid automaton to ensure that the chemical dosing remains in a safe range given the water level and flow rates. However, for now, we look at converting the simplified tank in Figure 1 to a hybrid automaton that can be represented in our verification tool. The hybrid automaton of the water tank is shown in Figure 2. It consists of two states:

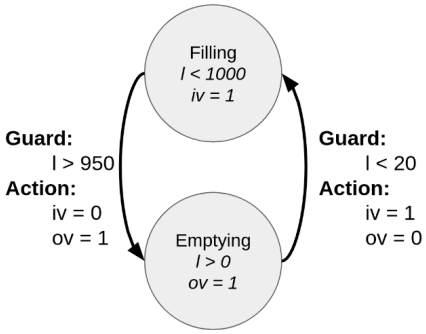


Figure 2. Conversion of Figure 1 into a simple automaton with guards, actions, and invariants between the two states.

`filling` and `emptying`. Filling has the invariants that the input valve is open ($iv = 1$) and the level is strictly less than the high maximum ($l < 1000$). Emptying has the invariants that the output valve is open ($ov = 1$) and the level is strictly greater than the low minimum. Note that the actual values of the thresholds and maximums/minimums are arbitrary and can be changed depending on the characteristics of the system under study. The automaton has the ability to change between the two states depending on the current water level in the tank. The system transitions from `filling`→`emptying` when the level is greater than 950 units by closing the input valve and opening the output valve (the 'Action' components in Figure 2). Similarly, the system transitions from `emptying`→`filling` when the water level is less than 20 units by closing the output valve and opening the input valve. Now with the system modeled as a hybrid automaton, we discuss our tool selection and the approach for modeling the system correctly.

3.2. Verification Tool

This work utilizes the C2E2 hybrid verification engine developed by the University of Illinois Urbana-Champaign. This tool was primarily selected due to the author's interest in working with the tool, but also because there are numerous examples available on its website that were helpful references. It has an easy to navigate user interface that allows for quick development and extension of automaton, simulation and verification of systems, and plotting of simulated results.

The project utilizes C2E2 v2.1.3 on Ubuntu 16.04.7 LTS. To ease the installation process, the software was installed using the VirtualBox Workstation image with the software preinstalled from the C2E2 webpage (2). Although the most recent version of the software was initially attempted, significant challenges with installation and dependency conflicts led to using an older version to save time. This image was imported into VirtualBox which instantiated the virtual machine and allowed for easy setup.

3.3. Iterative Approach & Ensuring Correctness

Finally, with the software installed and running, this section discusses the chosen approach to modeling the problem in C2E2 and how this supports functional correctness of the results.

We choose an iterative method to model the hybrid automaton from Figure 2 by starting with the basic water level monitors and adding complexities only after the system can be successfully simulated and verified. Specifically, we start with encoding the water level, then add the input and output valve states, and then the input/ouput flow rates. Once these are complete, we explore adding an extra state (`Idle`) and a timing component. By approaching the problem this way, it is decomposed into smaller, more manageable steps that can demonstrate the potential challenges of modeling such a system as a hybrid automaton in C2E2.

Additionally, the incremental advancements make debugging the model significantly easier. For example, in an early attempt to model the system, most of the control variables and transitions from the code base were directly encoded to start from a more advanced model. However, this resulted in numerous compounded errors that were difficult to reason about and resolve (especially given the poor supporting documentation). Once the model was simplified, the potential errors were limited and the results were easier to understand; leading to a more insightful verification process.

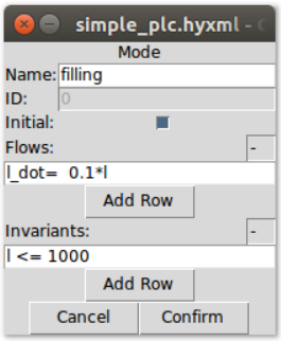
Finally, to ensure correctness of our C2E2 model (in addition to the iterative approach), we relied on C2E2's simulation tool to (1) check that potential runs of the model were safe, and (2) produce traces of different variables with respect to the model's runtime. Once the model was marked `Safe` by C2E2's simulation process, it was then verified in C2E2. This allows us to compare the output of the model with our expectations, as well as rely on C2E2's formal verification abilities—resulting in high confidence that our model is correct.

4. Results

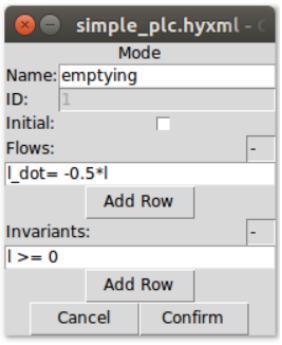
This section presents the results of modeling and verifying our hybrid model in C2E2. Each iteration includes screenshots of the the modes from C2E2 GUI, the initial set, the unsafe set for that requirement, and the resulting plots from C2E2's simulation output. Other specifications (such as transitions between states) can be found in the .hyxml files that are available at the previously linked GitHub.

4.1. Encoding Water Level

The hybrid model is initialized with two modes: (1) filling shown in Figure 3a and (2) emptying shown in Figure 3b. At this stage where no valves are added, the only invariant



(a) Filling mode.



(b) Emptying mode.

Figure 3. Filling and emptying modes of the C2E2 model.

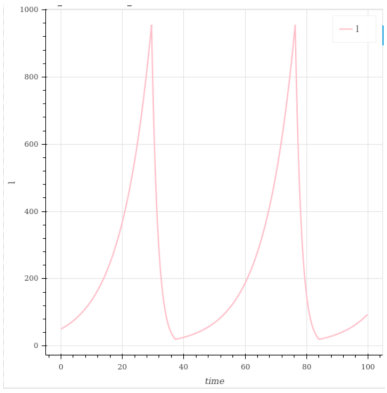


Figure 4. Water level of the model with starting level of 50 units.

added into C2E2 from Figure 2 is the level invariants. The flow equations are arbitrarily selected to appropriately show the tank either gaining water or losing water. With the modes and transitions now defined, the initial and unsafe set for this case need defined.

The initial set is decided to begin in the `filling` mode with the water level at 50 units and the unsafe set is defined when the level goes below 0 or above 1000. These are added to the C2E2 model as:

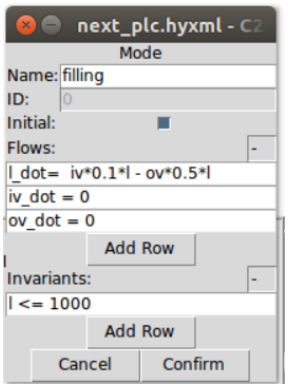
$$\textit{Initial Set: } \texttt{filling}\texttt{:}\ l == 50$$

$$\textit{Unsafe Set: } l < 0 \vee l > 1000$$

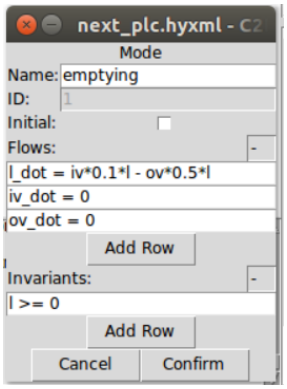
With the model complete and the initial and unsafe set defined, the C2E2 simulation engine is ran to determine if the model is safe. If the engine fails, either the model, the initial set, or unsafe set needs refined; if the engine returns 'safe', it is able to plot traces of the variables against the simulated time. The curve showing tank water level throughout the simulation is shown in Figure 4. As shown in the figure, the water level exponentially climbs until it reaches just shy of the maximum and then drops to just above the minimum; running in a cyclic pattern as one would expect. With the simulation complete, the model can now be verified by C2E2 by running the verification engine. In this instance, because the model and its transitions are extremely simple, the engine returned safe on the first attempt.

4.2. Adding Input/Output Valves

With the model correctly modeling the change in levels, the next step is to add the state of the input and output valves. This required introducing two more variables to the model: iv and ov. Additionally, the level flow is enhanced to account for the addition of these boolean variables; increasing the level when $iv == 1$ and decreasing the level when $ov == 1$. The updated modes are shown in Figure 5. Now, the initial set and unsafe set must be considered. For the initial set, in addition to the level starting at 50 units, the input valve must be high and the output valve must be closed.



(a) Filling mode.



(b) Emptying mode.

Figure 5. Filling and emptying modes of the next iteration of the C2E2 model.

$$\textit{Initial Set: } \texttt{filling}\texttt{:}\ l == 50\ \&\&\ iv == 1\ \&\&\ ov == 0$$

Another unsafe set is defined to ensure that the input valve and output valve are never opened at the same time. This is an example of a safety requirement that is highly dependent on the system operators and utility under study, but is easily encoded as an unsafe set in C2E2 as:

$$\textit{Unsafe Set: } iv == 1\ \&\&\ ov == 1$$

Finally, with the model appropriately defined, the simulation engine can be ran and the results can be plotted. In this case, the plot in Figure 6 shows the change of valve states for the input and output valves. As the equation governing the water level was not updated in this model, the plot of level vs. time is excluded for brevity. Similar to the previous section, as the additions to the model were simple, the verification engine was able to verify the system as safe on the first attempt.

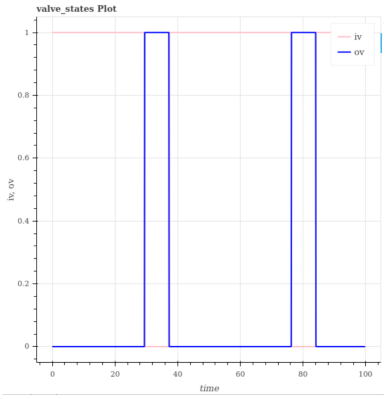
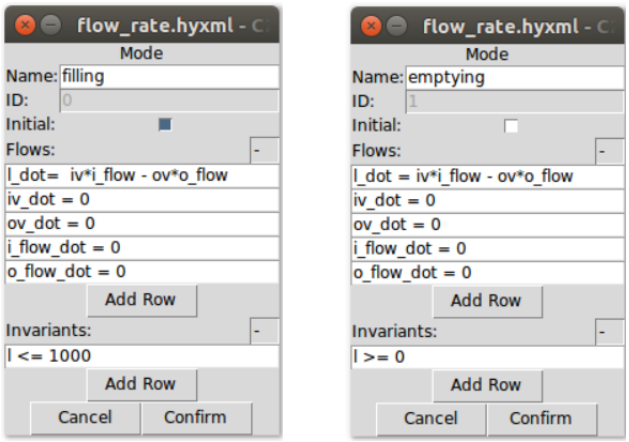


Figure 6. Input valve (red) and output valve (blue) states of the next iteration of the C2E2 model.



(a) Filling phase (b) Emptying phase

Figure 7. Filling and emptying modes of the model.

4.3. Adding Flow Rates

The next two sections explore the addition of variables that may complicate the overall system and enable the exploration of more interesting verification problems. We first introduce variable flow rates for the input and output valves; effectively adding two additional variables to our model and two additional flows to our modes (see Figure 7a and Figure 7b). The flow rates are initialized in the initial set shown below and may be updated when the model transitions from one state to the other. For our purposes, the flow rate was not varied from the initialized value when transitioning modes, but could easily be updated based on the current level if desired. Here, we initialize the input flow rate to be 0.3 and the output flow rate to be 0.0:

$$\textit{Initial Set: } \texttt{filling}\texttt{:}\ l == 50\ \&\&\ iv == 1\ \&\&\ ov == 0\ \&\&\ i\_flow == 0.3\ \&\&\ o\_flow == 0.0$$

Next, we consider an unsafe set in terms of the input flow rate and the output flow rate. When considering such a set, it is helpful to consider the real-world application; in this case, operators may prefer the tank to be empty rather than to overflow. In terms of flow rates, we want to ensure that the input flow rate is always strictly less than the output flow rate which can be written in C2E2 as:

$$\textit{Unsafe Set: } i\_flow - o\_flow >= 0$$

With the modes and sets properly defined, the simulation engine is ran and two traces are plotted. The first, shown in Figure 8, shows the relationship between each flow rate and the average flow rate as time progresses. The second, shown in Figure 9, shows the water level and average flow rate on the same graph. This enables us to check that the model is behaving properly as we expect the edges of each trace to align. Following successful simulation, the model is successfully verified using the C2E2 verification engine.

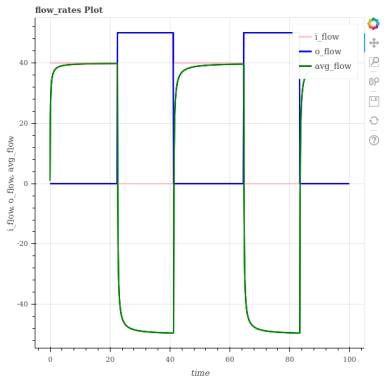


Figure 8. Plot of input flow rate (red), output flow rate (blue), and average flow rate (green) of the next iteration of the C2E2 model.

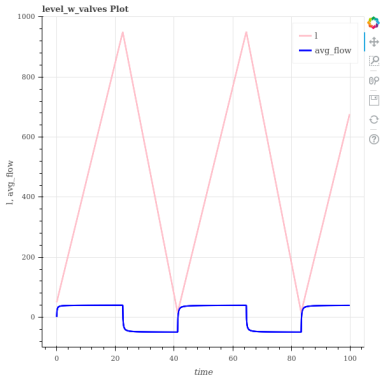


Figure 9. Plot of the water level (red) and average flow rate (blue) over time.

4.4. Adding an Additional 'Idle' State

Lastly, we expand the model to include a third `Idle` state where water is neither entering or leaving the tank. The

main addition to the model in this case is a `t` variable that tracks the time in discrete updates as shown in Figure 11. An updated hybrid automaton model of the system is shown in Figure 10. The initial set is expanded to include the time variable that is initialized to $t == 0.1$. This time was selected as earlier iterations experienced challenges setting the initial time to zero, but for the purpose of simulation/verification, this change is trivial.

$$\textit{Initial Set: } \texttt{filling}\texttt{:}\ l == 50\ \&\&\ iv == 1\ \&\&\ ov == 0\ \&\&\ i\_flow == 0.3\ \&\&\ o\_flow == 0\ \&\&\ avg\_flow == 1\ \&\&\ t == 0.1$$

The unsafe set for this case is defined to ensure the system progresses and does not stay in a single state too long. The opposite may also be checked to ensure the system does not change the state of the pumps too frequently and cause physical damage to the equipment.

$$\textit{Unsafe Set: } t > 20.0$$

With the model and sets complete, the simulation and verification process was ran for this case. The system returned safe for the simulation and verified for the verification. Graphs are omitted in this case as the timing constraint was generally more helpful to determine if the flow rates were appropriate and did not yield helpful graphs.

5. Discussion

In general, the results show promise for providing a helpful platform to ensure safe operation of a waste water treatment process. Each successive iteration of the model highlights how formal verification can systematically scale to include more realistic system variables and control logic. The ability to verify not just the water level, but also valve states, flow rate constraints, and timing behavior demonstrates that even modest extensions to the model can significantly enhance the utility of the verification framework. The cyclic plots

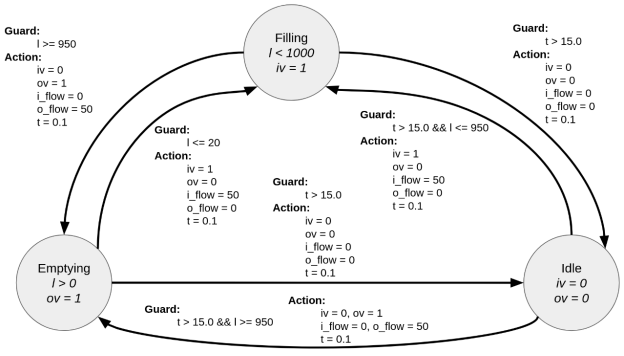
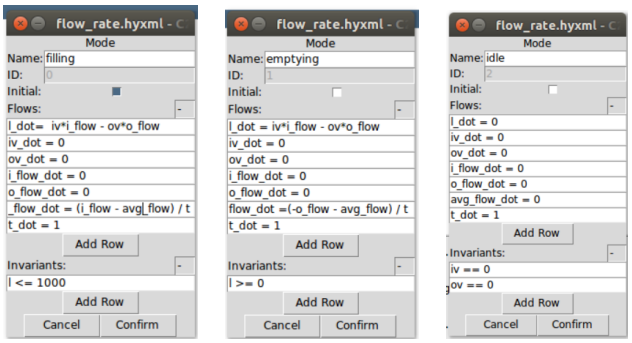


Figure 10. Hybrid automaton model with guards, actions, and invariants for the three states of the system.



(a) Filling mode. (b) Emptying mode. (c) Idle mode.

Figure 11. Filling, emptying, idle modes of the model.

of system variables confirm that transitions occur logically and within bounds, and unsafe conditions—such as both valves being open simultaneously—can be encoded and automatically checked. This supports the idea that verification tools like C2E2 can be integrated early in the control system design pipeline to preemptively identify design flaws before deployment.

In particular, subsection 4.3 and subsection 4.4 apprear able to help operators verify that their system design parameters will not lead to unsafe operation. For example, creating an unsafe set where time cannot be less than a certain threshold would ensure their design choices do not cause the valves to open and close to quickly (causing mechanical stress or failure). While starting from a simple system like we did ensures confidence in the overall results, the more complex hybrid automaton (Figure 10) allows for more robust exploration of safe and unsafe sets.

6. Limitations & Future Work

The scope of the project was significantly limited due to the difficulty implementing the system in C2E2. Throughout the project, and especially when expanding the system, there were numerous errors that C2E2 did not explain well and the supporting documentation did not address. Outside of difficulties using the tool, the properties verified are extremely basic and ideally would be more nuanced and interconnected to allow for more helpful insights from the verification process. Future work in this area should explore the addition of a chemical doping mechanism; a process typically found in waste water treatment processes and an extremely unsafe component if misconfigured. Additionally, the current implementation should be configured to be modeled after a real-world case study in an effort to shows its efficacy. It would be helpful to show how formal verification in this space is more useful during the design phase, rather than just before deployment.

References

[1] Sebastian Biallas, Jörg Brauer, and Stefan Kowalewski. Arcade.plc: a verification platform for programmable logic controllers. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 338–341, 2012.

[2] Lucas Brown, Chuchu Fan, Parasara Sridhar Duggirala, Suket Karnawat, Yu Meng, Sayan Mitra, Matthew Potok, Bolun Qi, and Mahesh Viswanathan. *C2E2 User's Guide Version 2.0*. University of Illinois at Urbana-Champaign, 2018. Available at https://publish.illinois.edu/c2e2-tool/files/2019/05/C2E2Manual_ver0.pdf.

[3] Simon Chadwick, Phillip James, Markus Roggenbach, and Tom Wetner. Formal methods for industrial interlocking verification. In *2018 International Conference on Intelligent Rail Transportation (ICIRT)*, pages 1–5, 2018.

[4] Borja Fernández Adiego, Dániel Darvas, Enrique Blanco Viñuela, Jean-Charles Tournier, Simon Bliudze, Jan Olaf Blech, and Víctor Manuel González Suárez. Applying model checking to industrial-sized plc programs. *IEEE Transactions on Industrial Informatics*, 11(6):1400–1410, 2015.

[5] Luis Garcia, Saman Zonouz, Dong Wei, and Leandro Pfleger de Aguiar. Detecting plc control corruption via on-device runtime verification. In *2016 Resilience Week (RWS)*, pages 67–72, 2016.

[6] Antonio Iacobelli, Lorenzo Rinieri, Andrea Melis, Amir Al Sadi, Marco Prandini, and Franco Callegati. Detection of ladder logic bombs in plc control programs: an architecture based on formal verification. In *2024 IEEE 7th International Conference on Industrial Cyber-Physical Systems (ICPS)*, pages 1–7, 2024.

[7] Sam Kottler, Mehdy Khayamy, Syed Rafay Hasan, and Omar Elkeelany. Formal verification of ladder logic programs using nusmv. In *SoutheastCon 2017*, pages 1–5, 2017.

[8] Soraya Mesli-Kesraoui, Olga Goubali, Djamal Kesraoui, Ibtihal Eloumami, and Flavio Oquendo. Formal verification of the race condition vulnerability in ladder programs. In *2020 IEEE Conference on Control Technology and Applications (CCTA)*, pages 892–897, 2020.

[9] Andreas Ulrich and Anjelika Votintseva. Experience report: Formal verification and testing in the development of embedded software. In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, pages 293–302, 2015.

[10] David Urbina, Jairo Giraldo, Nils Ole Tippenhauer, and Alvaro Cardenas. Attacking fieldbus communications in ics: Applications to the swat testbed. In Aditya Mathur and Abhik Roychoudhury, editors, *Proceedings of the Singapore Cyber-Security Conference (SG-CRC)*, pages 75–89. IOS Press, 2016.

[11] Mu Zhang, Chien-Ying Chen, Bin-Chou Kao, Yassine Qamsane, Yuru Shao, Yikai Lin, Elaine Shi, Sibin Mohan, Kira Barton, James Moyne, and Z. Morley Mao. Towards automated safety vetting of plc code in real-world plants. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 522–538, 2019.

[12] Xuankai Zhang, Jianhua Li, Jun Wu, Guoxing Chen, Yan Meng, Haojin Zhu, and Xiaosong Zhang. Binary-level formal verification based automatic security ensurement for plc in industrial iot. *IEEE Transactions on Dependable and Secure Computing*, pages 1–16, 2024.