Benchmarks for the Verification of Safety and Security Properties of PLC Programs in Cooperative Verification Environments

Chibuzo Ukegbu*
Hoda Mehrpouyan*
chibuzoukegbu@u.boisestate.edu
hodamehrpouyan@boisestate.edu
Boise State University
Boise, Idaho, USA

ABSTRACT

Cooperative verification is a promising technique for verifying the safety and security properties of Programmable Logic Controllers (PLCs). However, cooperative verification has not yet been widely used for PLC programs, and there are no cooperative verification benchmarks for PLC programs. This paper presents verification tools and property benchmarks for cooperative verification of the safety and security properties of PLC programs. We developed two algorithms: 1) to select and rank verification tools that satisfy the tool selection requirement of PLC programs. 2) A verifier validator that cooperatively verifies PLC programs using three complementary verification tools. We added a property benchmark that adds the end-of-the-cycle (EoC) variable to the LTL specification property to adjust the tool's verification from step semantics to the PLC's scan-cycle semantics. We conducted experiments using CoVeriTeam(a cooperative verification framework) with 40 real-world PLC programs from PLCOpen. Our approach significantly enhances the accuracy and reliability of PLC program verification by recommending tools that our experiments show to have the lowest false positive rate (FPR) and false negative rates(FNR). We recommend CBMC, CPA-SEQ, and Symbiotic as the go-to tools for cooperative verification of PLC programs using CoVeriTeam.

CCS CONCEPTS

• **Computer systems organization → Real-time system specification**.

KEYWORDS

PLC, CoVeriTeam, Benchmark, Verification,safety and security properties

ACM Reference Format:
Chibuzo Ukegbu and Hoda Mehrpouyan. 2023. Benchmarks for the Verification of Safety and Security Properties of PLC Programs in Cooperative Verification Environments. In *2023 8th International Conference on Information Systems Engineering (ICISE 2023), December 16–18, 2023, Bangkok, Thailand.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3641032.3641046

*Both authors contributed equally to this research.

ICISE 2023, December 16–18, 2023, Bangkok, Thailand

ACM ISBN 979-8-4007-0917-3/23/12
https://doi.org/10.1145/3641032.3641046

1 INTRODUCTION

Programmable logic controllers (PLCs) are widely used to control critical infrastructure systems, such as water treatment plants, nuclear power plants, and transportation systems. However, verifying the safety and security of PLC programs is a challenging task due to the heterogeneity of PLC architectures, control logic designs, the complexity of PLC scan cycles, and interrupt handling. Existing verification tools for PLC programs such as Cpachecker [9], cbmc [21], esbmc [18], FDBverifier [20], assume-guarantee reasoning [23–25] etc., often suffer from false positives (pointing to nonexistent property violations), false negatives (reporting no violations where there are violations) due to verification algorithm designs for software programs rather than PLC, such as overapproximation and underapproximation of time and process values to abstract the details from the state space to get a manageable or controllable set. False positives can cause unnecessary downtime and rework, while false negatives can lead to serious safety and security hazards. To curb these problems, false positives and negatives, combining the strengths of two or more tools became necessary. Cooperative verification is a technique that combines the strengths of multiple verification tools to improve the accuracy and reliability of verification results [6]. However, cooperative verification has not yet been widely used for PLC programs and, indeed, other conventional programs. The work available in the literature dealing with cooperative verification [6–8, 11] was not focused on safety-critical systems, except [6], which explains how effective the application of cooperative verification could be in safety-critical systems; however, no practical application or experiment was performed to establish benchmarks to guide tool selection, performance (verification time and scalability), and property specification to verify the safety and security properties of PLC programs. To the best of our knowledge, there is no investigation with respect to cooperative verification and setting the necessary benchmarks in programs, property, and tool selection to verify PLC logic. Therefore, this paper presents new tools and techniques as benchmarks for cooperative verification of the safety and security properties of PLC programs.

Our contribution is to develop a framework and benchmarks for cooperative verification of the safety and security properties of PLC programs. To this end, we developed algorithms and techniques that help to achieve the benchmarks in the following.

(1) A tool selection algorithm that selects three complementary verification tools from a cooperative verification environment based on user requirements and tool features.
(2) A property benchmark that adds the end-of-the-cycle (EoC) variable to the LTL specification property to adjust the tool's verification from step semantics to the PLC's scan-cycle semantics.
(3) A three-tool verifier validator that combines three verification tools selected by the tool selector algorithm to verify and validate PLC programs using the CoVeriTeam cooperative verification platform.
(4) Link to the repository containing our experiment artifacts for reproducibility purposes.

In addition to our contributions, our experiments serve as valuable experience reports, providing insight for fellow researchers entering the field of cooperative verification of PLC programs as research in this area continues to expand. We conducted experiments that involved our algorithms implemented for tool selection within a cooperative verification environment, which led to the selection of tools based on specific property requirements and tool features. Our custom verifier validator, designed to combine three different tools, is used with CoVeriTeam as a cooperative verification platform, confirming the capacity of the selected tools to detect violations through the synergy of their strengths. We evaluated the results using evaluation criteria, including false positive and negative rates (FPR and FNR) by establishing ground truths using PLCVerif [15] (a tool specializing in verifying PLC programs in their native language Structured Text (ST)), ultimately determining the most effective benchmark tools.

We further validated our findings by validating our results with PLCVerif and examining the result logs of the verification tools to uncover false positive and negative results. Our evaluation included 40 real-world PLC programs obtained from PLC Open and annotated PLC programs from [12]. Our results and experiences collectively confirm that our approach significantly improves the accuracy and reliability of PLC program verification. Furthermore, our recommended tools and property benchmarks provide valuable guidance for technicians and engineers who are navigating the cooperative verification of PLC programs. The subsequent sections of the paper are structured as follows: related work reviewed prior literature in the field, specifically focusing on benchmark development and cooperative verification. Our methodology section details the algorithms and techniques we have introduced. We proceed to present our case study, in which we describe the experiments carried out with the CoVeriTeam framework as a central element. Following this, the results and discussion section elaborates on our findings, and the results validation section explains how we verified these outcomes. Finally, our conclusion discusses the key takeaways, outlines the study's limitations, and hints at potential avenues for future work.

2 RELATED WORK

In this related work section, we explore two crucial facets: benchmark development within Industrial Control Systems (ICS) and the application of cooperative verification techniques to detect violations in Programmable Logic Controller (PLC) programs. This paper focuses on 1) addressing the creation of benchmarks and 2) examining the influence of cooperative verification on the safety and security aspects of PLC programs. To begin, we dive into the benchmark development literature before delving into the implications of cooperative verification for PLC program properties. First, within the domain of benchmark development, Althoff (2022) [3] presents a methodological approach explicitly tailored for the verification of the power system. This methodology systematically selects the types of verification problem, case descriptions, and property definitions. In particular, it leverages the CORA toolbox to generate various benchmarks to evaluate reachability analysis tools. However, it predominantly targets power systems, and its adaptability to domains like PLCs warrants further exploration. Beyer et al. (2019) [10] make significant contributions to benchmarking, focusing on automatic solvers and verifiers. Their framework outlines fundamental requirements, including resource measurement accuracy and reliable process termination. Although this framework enhances benchmarking practices, it primarily concerns the rigorous assessment of tools and the presentation of results, rather than the explicit development of cooperative verification benchmarks for PLC programs. On the contrary, our paper is dedicated to optimizing cooperative verification platforms for the assessment of the PLC program, encompassing the critical aspects of tool selection, specification properties, and test case creation. Transitioning to cooperative verification's influence on PLC program verification, recent works, including those by Beyer et al. [6–8, 11, 12], underscore the transformative potential of cooperative verification, especially in the context of safety-critical infrastructures. Cooperative verification offers promise but requires various tools, properties, and program benchmarks customized for varied cooperative verification environments. Our research contributes significantly to this endeavor, providing practical tools and techniques to assist practitioners and researchers in adeptly navigating the cooperative verification of PLC programs, mainly through platforms like CoVeriTeam.

3 METHODOLOGY

As depicted in figure 1, the proposed methodology aligns with those discussed in the related work section (cf. Section 2), although tailored to the unique behaviors of PLCs.

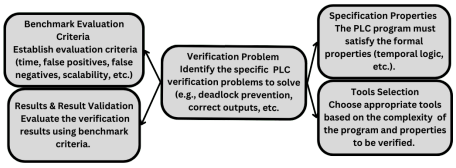

Figure 1: Steps in the Methodology

The approach begins with carefully selecting verification problems tailored to PLC program behaviors. A tool selection criterion is established to choose appropriate tools that align with the PLC program scan cycle semantics and enable adequate cooperative verification. Specification property requirements are defined to outline desired properties for PLC programs, accompanied by criteria for evaluating proposed benchmarks. Key evaluation metrics include false positives, false negatives, false positive rate (FPR), false negative rate (FNR), and tool execution time thresholds. The methodology ensures the integrity of the results through an incorporated validation step, further enhancing the precision of the experimental findings. We will proceed systematically, addressing each step in detail to fully explain each.

3.1 Problem Identification

Addressing the challenges associated with PLC programs requires a clear understanding of specific verification issues. These issues, as described in [14], cover safety properties, deadlock prevention, and ensuring correct output. In this paper, we narrow our focus to the safety and security of PLC programs, emphasizing the accuracy of the output when provided with specific inputs. This emphasis is rooted in the critical nature of industrial control systems, where deviations from the correct output can lead to safety violations and potential hazards in various sectors. We carefully select PLC programs with safety and security properties to achieve this, leveraging openPLC programs with specially designed annotations. This process involves identifying relevant properties, establishing selection methodologies for tools and properties, and ensuring alignment with the scan-cycle semantics of PLC programs, enabling adequate cooperative verification for accurate and comprehensive results.

3.2 Specification Property Requirement

Referring to Table 1, we have outlined the requirements for specification properties essential to verify PLC programs in a cooperative verification setting. These requirements are the foundation for our experiments in

assessing their compliance within the context of cooperative verification using the CoVeriTeam-compatible Linear Temporal Logic (LTL) dialect.

The table highlights the importance of representative properties that span safety and security to fully cover the domain of PLC systems. Invariance and reachability ensure correctness and safety, while safety properties are critical for Industrial Control Systems (ICS). Security properties are essential for data protection, and boundary properties are crucial for testing tool correctness. These requirements and rationales guide the selection and definition of properties for verification of the PLC program in cooperative environments.

Specification Property Requirement Table	
Requirement	**Rationale**
Observable state: Properties should be amenable to change so that extra variables that indicate the observable state of PLC programs can be added to the property specifications	PLC observable states must be identified for correct verification of PLC programs because the PLC program's end-of-cycle (EoC) is the observable state that helps balance the steps and cycle semantics of conventional IT programs and OT PLC logics.
Representative properties: Properties must represent the domain of PLC systems, from safety properties to security properties.	A good benchmark must cover the important areas of the domain of interest.
Invariance: This is a condition that remains true throughout the execution of the program.	Properties we test should remain true in all states to ensure correctness.
Reachability: Ability to reach a specific state from an initial state.	Specification properties should be able to reach any state of interest for safety reasons.
Safety: Ensuring that no hazard is created from violations of safety properties.	The lifeblood of ICS is the safety of the systems and personnel. Therefore, the safety properties should be used as benchmarks for the selection of the verification tool.
Security: The ability to ensure that there are no data leaks or access control violations within the program.	Security properties are necessary to determine the benchmark properties for the selection of cooperative verification tools.

Table 1: Property Specification Requirements for PLC Program's Property Benchmarks

In the context of Programmable Logic Controllers (PLCs) with their characteristic scan-cycle semantics, applying Linear Temporal Logic (LTL) properties tailored for conventional programming languages like C and C++ poses a challenge, which is processing PLC programs in step semantics instead of cycle-semantics. We introduce the End-of-Cycle variable (EoC) concept to adapt the properties of LTL to PLCs, as discussed by Bohlender et al. [12]. For example, consider an LTL specification property formula like: $G(\text{EmergencyStop} \rightarrow O(\text{OperationStopped}))$ Here, "EmergencyStop" represents the condition when the emergency stop button is pressed, and the specification asserts that it will eventually lead to "OperationStopped" becoming true immediately or at some point in the future. By introducing the EoC variable, we can transform the formula into the following. $G(\text{EmergencyStop} \rightarrow (EoC \cup O(\text{OperationStopped})))$ In this modified formula: "EoC" represents the end of a cycle in the PLC operation. "EmergencyStop" and "OperationStopped" are specific conditions or events within the system. The "G" operator ensures that the entire expression is globally valid across all time steps. The "→" operator denotes an implication. The "O" operator indicates that the following condition will hold at some point in the future. The "U" operator signifies the "Until" operator, which asserts that the left condition holds until the proper condition becomes true. This adjustment accommodates the PLC scan-cycle semantics, enabling precise specification and verification of system behavior.

3.3 Tool Selection Requirement

Building upon the framework proposedby [22], which outlines selecting effective security vulnerability and testing (SVT) tools over trivial ones by providing specific requirements, our tool selection process carefully considers specific criteria. In cooperative verification environments, many tools are encountered, each characterized by various types and intended verification purposes. Moreover, the behavior of Programmable Logic Controller (PLC) programs sets them apart from other programming paradigms like C or Java. Consequently, we have outlined the requirements for any tool we choose.

Tool Requirements Table	
Requirement	**Rationale**
TR1: Modular Verification tool (can run more than one algorithm)	Modular tools provide the opportunity to try more than one verification method (such as predicate abstraction, abstract interpretation, symbolic execution, etc.), which makes for robust verification.
TR2: Verification tool that can verify real-time property specifications	PLC programs' behaviors are mostly real-time, and a tool that recognizes that.
TR3: Flexibility: Flexible Verification tools can run different algorithms not designed for a specific verification algorithm	Flexible tools make for easy cooperative combination. And can verify both safety and security properties.
TR4: Machine-readable output	Machine-readable output fosters post-verification analysis and validation.
TR5: Verify C programs or Structured Text PLC programming language	The tools have to verify C programs or ST PLC programs because, in our experiment, we converted ST to C for our experiments.
TR6: Memory safety verification	PLCs have little memory; therefore, the safety of the memory from memory violations is necessary.
TR7: Support for LTL and symbolic model checking	The properties to be verified are written in LTL dialect in CoVeriTeam, and symbolic model checking increases fast analysis of large and complex systems such as ICS.

Table 2: Tool Requirements Table

Table 2 outlines the critical criteria for selecting appropriate verification tools in cooperative verification environments, particularly for programable logic controller (PLC) programs. These requirements are carefully crafted to address the unique challenges of PLC programs, which exhibit real-time

behaviors distinct from traditional programming languages. The first requirement, Tool Requirement-1 (TR1), emphasizes the need for modular verification tools capable of employing various verification methods, enhancing the robustness of the verification process. TR2 underscores the importance of real-time property verification, aligning with the temporal nature of PLC program behaviors. TR3 highlights the importance of tool flexibility, allowing them to run diverse algorithms not limited to a specific verification technique, facilitating cooperative verification across safety and security properties. TR4 advocates for machine-readable tool outputs, which facilitate post-verification analysis and validation. TR5 necessitates the ability to verify both C and Structured Text PLC programs, accommodating the experimental conversion of Structured Text to C. TR6 emphasizes memory safety, while TR7 requires tools to support LTL and symbolic model checking. Collectively, these requirements serve as a comprehensive foundation for selecting tools that can effectively address the complexities of verifying PLC programs in cooperative environments. We will describe the tool selection algorithm and how it works.

Algorithm 1 serves the purpose of aiding in the selection of the most suitable tool from a list of candidates for the verification of the PLC program. The process begins with extracting tool-related data from an online database compiled from the tools' official documentation. These data form the basis for the evaluation of the tools. Each tool in the list is characterized by its characteristics and is assigned an initial score of zero. The scoring process is discerning, with scores weighted to reflect the relevance of each feature to the verification of the PLC program. In particular, features crucial to this task, such as real-time properties and safety, receive higher scores, while features such as open source nature are assigned lower scores.

The algorithm systematically assesses the alignment of each tool with predefined criteria, producing a ranked list of tools that best meet the criteria for the verification of the PLC program. A unique feature of this implementation is providing tool features and specifications directly within the script, bypassing the need for a real-time online database. In summary, Algorithm **??** streamlines the selection of the optimal tool for the verification of the PLC program by meticulously scoring the features of each tool according to their relevance. The resulting ranked list of tools aligns with the specific needs of PLC verification, with higher scores assigned to critical features in this context, ultimately facilitating the selection of the most appropriate tool for the task. The adaptability of this implementation, utilizing hard-coded features and specifications, ensures functionality even when a comprehensive online database is unavailable, maintaining the algorithm's efficacy.

Algorithm 1: Tool Selection Algorithm

Data: C - A set of criteria used for tool evaluation
T - A set of tools to be evaluated.
$M(c_i, t_j)$ - A function that determines whether a tool t_j meets the criteria c_i
D - An online database containing tool data
Result: The ranked list of tools based on how well they meet the criteria and the data from the online database

```
// Fetch tool data from the online database D
   and create a list L
```
(1) Fetch tool data from the online database D and create a list L;
(2) **forall** $t_j \in T$ **do**
```
    // Initialize the total score S(t_j) to 0
```
(3) Initialize the total score $S(t_j)$ to 0;
(4) **forall** $c_i \in C$ **do**
(5) **if** $M(c_i, t_j) = 1$ **then**
```
        // Add the criterion's score S(c_i) to
           the total score S(t_j)
```
(6) Add the criterion's score $S(c_i)$ to the total score $S(t_j)$;

```
// Rank the tools in L based on their total
   scores S(t_j)
```
(7) Rank the tools in L based on their total scores $S(t_j)$;
(8) **return** *The ranked list of tools*

3.4 Tools Evaluation Criteria

To evaluate the performance of the tools, we devised criteria to evaluate the performance of the tools to identify the benchmarks.

Table 3 summarizes the critical evaluation criteria for the verification tools of the PLC program, focusing on the vital aspects of the evaluation of the performance of the tool. It emphasizes the importance of false negatives and positives as critical metrics. Introduce a 30-minute execution time threshold to manage tool performance. Compatibility is critical to evaluating how tools can collaborate effectively in the verification process. Verification time is excluded from the criteria due to its unreliability, often influenced by external factors such as BenchExec and CPU throttling. The approach involves a two-step process to accurately distinguish false positives and false negatives from tool-generated results. The deliberate introduction of errors into select PLC programs creates ground truths for the accurate identification of violations. Subsequently, verification tools are executed on these modified programs and their results are compared with ground-truth data. False positives are identified when a tool incorrectly marks non-violations as violations, indicating unnecessary alerts. False negatives arise when a tool does not identify known violations, signifying a lack of sensitivity. The methodology, results, and validation are discussed in more detail in the forthcoming case study section.

Evaluation Criteria	
Requirement	**Rationale**
False negatives (FNs)	False negative means that the real violations are not detected.
False positives (FPs)	False positives show the ability of a tool to detect false violations.
Execution time threshold	The more time a tool takes to verify a PLC program, the BenchExec will terminate the process if it goes beyond the set threshold, which is set at 30 minutes.
Compatibility	When tools are combined, their ability to work together helps to know tools that could cooperate to verify PLC programs.
False Positive Rate	This helps us to know each tool's rate of false positives after several verifications.
False Negative Rate	This helps to find the rate of false negatives for each tool after several verifications.

Table 3: Tools' Evaluation Criteria Table

Our methodology comprises two core components: Methodological steps and methodological flow. Methodological steps systematically guide the benchmarking process, covering the definition of the verification problem, the specification properties, the selection of tools, the benchmark criteria, and the analysis of the results. In the Methodology Workflow, as illustrated

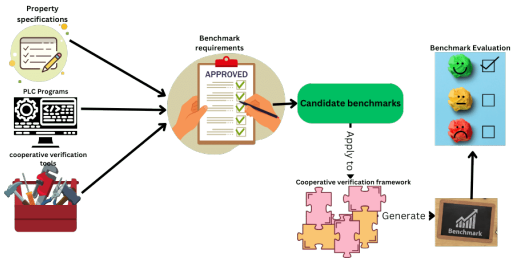

Figure 2: Proposed Benchmarking Process

in Figure 2, we outline critical requirements, including property, tool, and program selection, vital to defining candidate benchmarks tailored to the unique characteristics of PLC programs. The selected cooperative verification tools are then applied to these candidates, evaluating their performance based on established metrics like false positives, false negatives, and verification time. The results of this assessment yield two critical types of benchmarks: property specification benchmarks and cooperative verification tool benchmarks. These benchmarks are subject to further evaluation to ensure that they meet predefined criteria, adapting methodologies from related domains to suit the specific context of PLC program behavior, as referenced in earlier works [10], [16], and[22], respectively.

4 CASE STUDY

In this section, we present case study experiments, focusing on critical components of our workflow. We introduce CoVeriTeam, a cooperative verification tool, and discuss the specific PLC programs and specification properties to be verified. The experiments include the implementation of the tool selection algorithm, using a custom three-tool verifier validator, and the significance of this approach. We present the results obtained and follow with a discussion and validation, emphasizing their implications for Industrial Control Systems (ICS) security.

4.1 Case Study Tool - CoVeriTeam

Cooperative verification is a collaborative approach that involves multiple verification tools that work together to assess the accuracy and reliability of the system or program, with the objective of improving the effectiveness of the verification process by leveraging the strengths of different tools and mitigating their limitations (Beyer et al., 2022). CoVeriTeam orchestrates the collaboration of various verification tools, allowing them to jointly analyze program properties of the Programmable Logic Controller (PLC), detect potential violations, and optimize the verification process while saving time. The evaluation of the performance of the cooperative verification tool, the verification of the program, and the imposing of a 30-minute time limit for each verification process [5] are part of this case study. For more information on CoVeriTeam and its applications, readers can click here to visit the CoVeriTeam GitLab page.

4.1.1 Custom Verifier Validator. We developed a custom verifier validator using three verification tools to verify PLC programs due to the high safety-criticality of Industrial Control Systems (ICS). This tool provides an additional validation layer that includes a stringent protocol for handling errors and false or unknown verdicts, involving manual code review in conflicting cases(where the three tools have conflicting verdicts) to ensure safety and security standards are met. These tools are selected by our algorithm described in the methodology section (cf. 1). The critical security advantage is that if one tool fails, the other two with distinct verification attributes can detect violations, preventing false negatives that might occur with single-tool verification.

From Algorithm 2, we describe the process of creating the custom verifier validator as follows:

- Create actors: We create three "actors" (three verification tools) named verifier, validator1, and validator2 from the createActor object in the CoVeriTeam interpreted language.
- Define the conditions: We check whether a variable called "verdict" has the value "true" or "false" and store the result in two conditions, condition one and condition 2. Suppose that the first verifier's result is true or false. In that case, the second and third verifiers will validate the result by verifying the same PLC program using the results of the previous verifiers.
- Create and execute components: Based on these conditions, we create two components (parts of our program), a second-component, and a third component. These components are used to perform verification tasks using specified verifiers, and we run them.
- Print actor type: We check and print the type of one of the actors, which helps us to know its name and the execution technique (Sequence or Parallel).
- Print artifacts: Finally, we print the results or artifacts produced by our program when we execute the components. These artifacts are sent to a folder within CoVeriTeam called cvt-output that contains an XML verification result.

Algorithm 2: Custom Verifier Validator Algorithm

```
Create actors:
    verifier ← Create ProgramVerifier from YAML file at verifier_path
    validator1 ← Create ProgramValidator from YAML file at validator1_path
    validator2 ← Create ProgramValidator from YAML file at validator2_path
Define conditions:
    condition1 ← Check if verdict is TRUE or FALSE
    condition2 ← Check if verdict is TRUE or FALSE
Create and execute components:
    second_component ← ITE(condition1, validator1)
    third_component ← ITE(condition2, validator2)
Execute actors:
    res ← Execute verifier followed by second_component
    res1 ← Execute verifier followed by third_component
Print actor type:
    Print the type information of verifying_verifier
Print artifacts:
    Print res and res1
```

4.2 PLC programs and Properties Verified

4.2.1 PLC Programs Verified. The PLC programs that we used in the experiments are obtained from the PLCOpen safety library, and [12] annotated PLC programs. Our choice is based on the fact that most of the PLC programs have been used for an experiment and, as such, are accessible and reliable. The PLC programs cover important industrial processes in the ICS, such as EmergencyStop, DiagnosticConcepts, SafeMotionIO, and safeMotion. We describe the PLC programs below:

- EmergencyStop: This PLC program involves the implementation of an emergency stop or shutdown procedure within an industrial system. It may be designed to stop the operation of machinery or processes quickly and safely in a critical emergency to prevent accidents or damage.
- DiagnosticConcepts: Diagnostic Concepts is a PLC program related to diagnostics and monitoring. Collects data from sensors and devices in an industrial system, processes diagnostic information, provides insight into the health of the system, or identifies potential issues.
- SafeMotionIO: SafeMotionIO refers to the control of safe motion and motion-related functions in an industrial setting. This program focuses on ensuring safe operation of machinery, including aspects such as motion control, positioning, and collision avoidance, while adhering to safety standards and protocols.
- SafeMotion: Safe motion refers to motion control in an industrial environment, focusing on safety. It encompasses various safety features and functions associated with motion control systems, such as safe acceleration and deceleration, speed monitoring, and error handling.

These PLC programs, including "EmergencyStop," "DiagnosticConcepts," "SafeMotionIO," and "safeMotion," serve as valuable representatives of Industrial Control Systems (ICS) for benchmarking purposes. They encompass critical aspects of industrial automation, such as safety, security, diagnostics, and motion control, which are fundamental in various ICS applications. We use 40 of these programs, 25 single-module programs, and 15 multi-module programs. We want to ensure that the tools are evaluated in simple and complex PLC programs to ascertain their suitability and efficiency in verifying them.

4.2.2 Specification Properties Used For Verification. Our case study focused on verifying safety and security properties within PLC programs, with a particular emphasis on security due to its increasing relevance in today's Industrial Control Systems (ICS) exposed to external networks. Although safety properties were not exhaustively listed, they were annotated in the programs, meeting the requirements outlined in our methodology section. Security properties are paramount in ensuring secure ICS operation, and their assessment is intricate due to the unique characteristics of PLC programs.

4.2.3 Security Properties Considered. The aim is to identify and formulate security properties that accurately represent violations in programming logic controller (PLC) programs, focusing primarily on access control vulnerabilities. According to [28], PLC security vulnerabilities can be broadly classified into network, firmware, and access control issues. Our primary concern centers on access control and program modification attacks, as compromised access can potentially lead to tampering with the process logic. These security concerns will be converted into verifiable properties to effectively assess benchmark verification tools. Examples of security concerns in PLC programs, as identified by [27] and [1], include:

(1) Modification of Real-Time Inputs: Vulnerabilities arise when attackers manipulate the input of the real-time sensor.
(2) Multiple Assignments for Output Variables: Race conditions can occur when an output variable depends on multiple timers or counters, potentially leading to non-deterministic behavior.
(3) Uninitialized or Unused Variables: Unused input or output variables can be exploited by attackers to send malicious input or output, and uninitialized variables may receive malicious values at runtime.
(4) Improper Runtime: Inputs manipulated at run-time can induce runtime errors, introducing vulnerabilities.

To assess these security concerns, we express them as properties in Linear Temporal Logic (LTL) and use them to evaluate the capability of candidate benchmark verification tools in detecting PLC program security vulnerabilities. Additionally, we adapt the property specifications to align with PLCs' operational behavior and scan cycle semantics (cf. 3.2). This distinction helps to differentiate step semantics from PLC cycle semantics, enhancing the accuracy of benchmark tool evaluations in security verification.

4.3 Experiments

In this experiment, we set out to establish benchmark tools and properties for the cooperative verification of PLC (Programmable Logic Controller) programs within the CoVeriTeam framework, using CoVeriTeam as a case study cooperative verification framework. The process included multiple crucial steps: We started the experiment by implementing a Python-based tool selection algorithm that systematically evaluated specific features extracted from the CoVeriTeam tool library documentation. These features included safety and security property checks, real-time verification support, over-approximation, abstract interpretation capabilities, Counterexample-Guided Abstraction Refinement (CEGAR) integration, the presence of a graphical user interface (GUI), and open source status. The documentation of each tool was subjected to rigorous scrutiny for these characteristics, and each characteristic was assigned a weighted score. The cumulative scores for each tool allowed us to classify them according to their alignment with predefined tool selection requirements for the PLC programs, as detailed in Table 2. Due to constraints related to developing an online verification tool's feature database/ontology, we directly integrated the required features into the Python script. The script autonomously

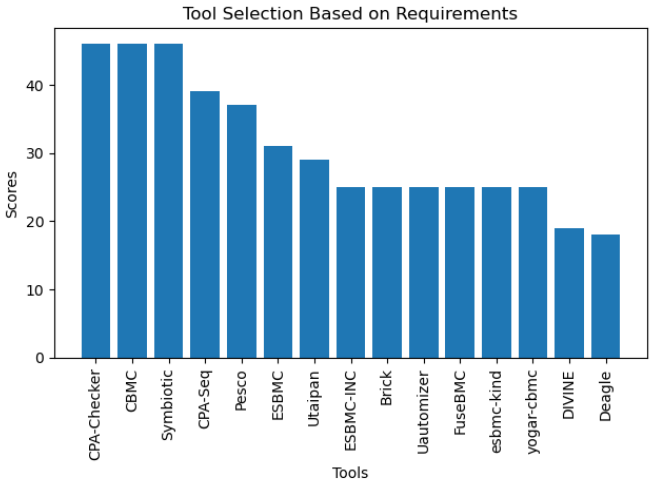

Figure 3: Verification Tools Selection Ranking in CoVeriTeam

selected tools that demonstrated compatibility with the behavioral characteristics of the PLC programs. As shown in Figure 3, the results were visually presented, highlighting the top-ranked tools recommended for the custom verifier validater. The custom verifier validator is crucial in our experiment, allowing us to assess the performance of the three selected tools and determine how effectively they detect actual violations. Specifically, we used the custom verifier validator to accommodate the trio of tools chosen by the tool selection algorithm. This strategic approach allowed us to subject these tools to rigorous tests to identify the most suitable options to verify PLC programs while meeting our stringent evaluation criteria. Our aim is to discern the tools that collectively provide cooperative verification with zero false negatives or false positives. This process is a pivotal step in determining the ideal combination of tools to effectively verify PLC programs, ensuring accuracy and reliability in our results.

Our experimental approach involved selecting sets of three tools for each round based on safety and security properties, ensuring that the chosen tools consistently ranked high according to the recommendations of our selection algorithm, as shown in Figure 3. In the category of highest ranking, we selected Symbiotic, Pesco [26], CBMC [21], Cpachecker [9], ESBMC-incr (selected ahead of ESBMC for its incremental algorithm) and Cpa-seq. Additionally, we intentionally integrated tools that were not prioritized by the selection algorithm to assess their performance. We included FuseBMC [2], Brick [13], Symbiotic, and Cpachecker, Brick and FuseBMC; The next rounds involved tool sets such as ESBMC-Kind [17], Yogar-cbmc [29], Cpa-seq, Divine [4], Deagle [19], and ESBMC-incr.

In the subsequent experiment, our aim was to assess the authenticity of the benchmark tools by calculating their false-positive and false-negative rates. To achieve this, we deliberately introduced violations into the PLC programs and relied on an external tool, PLCVerif, to establish ground truths. We then selected the top three performing tools and combined them using a custom verifier validator to verify 13 multimodule PLC programs. The verification time was extended from 30 to 60 minutes to gauge potential improvements. This approach provided a robust evaluation of the effectiveness of the tools, particularly for programs that had previously returned unknown or canceled. We have made all of the experimental artifacts available in a repository for transparency and reproducibility. These experiments were carried out on a Linux Ubuntu 22.04 system with specific hardware specifications.

4.4 Results and Discussion

The tables, as in Table 4 and Table 5, show the results of various verification tools compared with the ground truth established by PLCVerif. This ground truth was determined through deliberate violations, ensuring the correctness. Our result evaluation criteria helped calculate false positives, false negatives, true positives, and the respective rates, as summarized in Table 8, with "True" indicating absence, and "False" indicating presence of violations, "Unknown" means inconclusive due to timeout or complexity; an "error" points to inability of the verification tool to successfully analyze the PLC program, due to issues relating to compatibility to the PLC program behavior.

Table 8 summarizes the performance of various verification tools in identifying security and safety issues. Some tools, such as CPA-SEQ and ESBMC-INCR, show low False Negative Rates (FNR), indicating their effectiveness in recognizing critical problems. However, ESBMC-INCR encountered challenges, generating numerous unknown results due to time constraints and CPU throttling. In contrast, tools such as CPA-SEQ and Symbiotic demonstrate low False Positive Rates (FPR), reducing false alarms and supporting operational efficiency. The discussion does not separately address True Positives (TP) and True Negatives (TN), as they contribute to the FPR and FNR calculations. Balancing FNR and FPR is vital. Tools that excel in one area may lack in the other, necessitating a trade-off. High FPR tools hinder efficiency, while high FNR tools risk undetected issues, especially in critical applications. Low-FPR and FNR tools can produce better results, as shown in Tables 6 and 7. The use of strengths and the extension of the verification time improved accuracy for many PLC programs, benefiting programs 9, 10, 12, 13, 17, 19, 22, 23, 25, 28, 32, 33, and 38. Verification time was not a primary factor in our analysis due to the unreliability of ICS critical to safety, where precision and reliability were paramount.

4.5 Brief Experience Report

Our CoVeriTeam experiments uncovered limitations, notably the platform's requirement for property files with the a.prp extension, which triggered exceptions when we attempted compliance. Adherence to the specialized Linear Temporal Logic (LTL) dialect is crucial, especially with the introduction of the "end-of-cycle" (EoC) variable for cycle semantics alignment. We found that single-module PLC programs with simple function blocks were more accessible for verification than multimodule programs. In particular, symbiotic, CBMC, and CPA-SEQ emerged as preferred cooperative PLC program verification choices through CoVeriTeam.

4.6 Result Validation With PLCVerif

To validate our results and create a reliable benchmark, we turned to PLCVerif, a specialized tool to verify PLC programs in their original Structured Text Language (ST). Although the verification tools were initially designed for C programs, we first translated the ST programs into C to align with them. However, PLCVerif's native ST language support stood out for its efficiency, as it is more reliable due to its post-verification abilities and clear results that show whether there was an error or none. During validation, we intentionally introduced violations into the 40 PLC programs, leveraging PLCVerif's capabilities beyond primary verification, such as model reduction and support for expressing requirements informally in everyday language. This inclusive validation process ensures the credibility of our findings, enabling a confident evaluation of the performance of the benchmark tool. In summary, PLCVerif played a crucial role in validating our results, allowing us to establish a reliable benchmark for our study's candidate tools. It demonstrated its strength in handling PLC programs in their original language. It offered additional features that improved the assessment of property formats, ensuring the robustness and credibility of our findings.

Table 4: Verification Result Table for Tools Recommended by the Tool Selector Algorithm

#	GroundTruth	Symbiotic	Pesco	Cbmc	Time	#	GroundTruth	Symbiotic	Pesco	Cbmc	Time
1	True	True	true	True	0.371	21	False	True	False	true	0.74
2	True	True	true	True	0.367	22	True	True	unknown	True	4.33
3	False	True	false	false	1.345	23	True	False	unknown	Error	10.75
4	True	True	True	unknown	4.34	24	False	True	true	True	0.56
5	True	False	False	unknown	5.33	25	True	True	True	true	0.82
6	False	True	True	false	2.72	26	True	True	unknown	unknown	28.6
7	False	True	unknown	True	3.54	27	Unknown	True	Error	unknown	25.3
8	True	True	True	True	0.93	28	False	True	True	True	1.92
9	True	True	unknown	unknown	20.22	29	True	True	False	true	1.10
10	False	True	True	false	1.54	30	True	True	False	False	2.22
11	True	True	True	True	0.65	31	True	True	True	True	0.45
12	False	True	False	True	0.90	32	True	False	False	True	1.45
13	False	True	unknown	Error	15.30	33	False	True	True	unknown	6.63
14	True	True	True	true	0.42	34	False	True	True	False	1.34
15	True	True	unknown	True	3.35	35	False	True	True	True	1.52
16	True	True	False	False	2.23	36	False	True	True	unknown	7.54
17	False	True	False	True	1.12	37	True	True	False	False	2.12
18	False	True	True	unknown	5.59	38	False	True	Timeout	unknown	25.31
19	True	True	True	True	0.67	39	False	False	True	True	0.651
20	True	True	True	False	2.21	40	True	True	True	True	0.542

Table 5: Verification Result Table for Tools Recommended by the Tool Selector Algorithm

#	GroundTruth	Cpachecker	esbmc-incr	-seq	Time	#	GroundTruth	Cpachecker	esbmc-incr	cpa-seq	Time
1	True	True	True	True	2.519	21	False	True	Timeout	True	29.32
2	True	Unknown	True	Unknown	4.67	22	True	True	Unknown	Timeout	2.71
3	False	True	Unknown	True	3.46	23	True	True	Unknown	Unknown	28.58
4	True	True	True	True	5.057	24	False	True	Unknown	True	2.72
5	True	True	False	True	28.60	25	True	True	Unknown	Unknown	4.96
6	False	True	True	True	2.39	26	True	True	Unknown	True	2.87
7	False	True	True	Unknown	12.04	27	Unknown	True	Unknown	Unknown	29.14
8	True	True	Timeout	True	2.63	28	False	True	Unknown	Unknown	2.464
9	True	True	Unknown	Unknown	24.16	29	True	True	Unknown	True	4.24
10	False	True	Unknown	Unknown	7.97	30	True	True	Unknown	True	25.12
11	True	True	Timeout	True	3.55	31	True	False	True	Unknown	2.34
12	False	True	Unknown	Unknown	2.53	32	True	Unknown	False	Unknown	1.57
13	False	True	Unknown	True	24.45	33	False	True	Unknown	Unknown	4.27
14	True	True	Unknown	True	3.33	34	False	True	Unknown	False	29.12
15	True	True	Unknown	True	5.14	35	False	False	Unknown	False	28.56
16	True	True	Unknown	True	29.86	36	False	True	False	True	6.23
17	False	True	Unknown	Unknown	2.52	37	True	True	Unknown	True	29.67
18	False	True	Unknown	True	2.49	38	False	True	Unknown	Unknown	4.08
19	True	True	Unknown	Unknown	29.76	39	False	True	False	True	2.88
20	True	True	Unknown	True	29.74	40	True	False	Unknown	True	2.77

5 CONCLUSION, LIMITATIONS AND FUTURE WORK

In conclusion, this paper addresses the critical challenge of ensuring the verification of safety and security for PLC programs by introducing innovative tools and techniques to serve as benchmarks for cooperative verification. We have devised a tool selection algorithm that ranks verification tools based on their features and adapts specification property formulas to suit the behavior of the PLC program. Our custom verifier validater improves the accuracy and reliability of the verification of the PLC program, helping technicians and engineers navigate this complex domain. We have developed benchmark verification tools optimized for CoVeriTeam and validated our results using PLCVerif. Although the current implementation includes tools within the script, future work will focus on establishing an online database or ontology of verification tools for more efficient tool selection. We plan to refine and expand our benchmarks to cover a wider range of safety

Table 6: Verification Results for Top Performing Tools

#	GT	Symbiotic	CBMC	CPA-SEQ	Time (min)
1	T	T	T	T	0.323
2	T	T	T	T	1.12
3	F	F	F	T	0.83
4	T	T	T	T	0.74
5	T	T	T	F	1.29
6	F	F	unknown	F	20.32
7	T	T	unknown	T	29.9
8	T	T	T	unknown	23.1
9	F	F	F	F	28.2
10	F	T	F	F	2.63
11	T	T	T	T	1.39
12	T	T	F	unknown	1.88
13	F	T	F	T	4.62

Table 7: Verification Results and Evaluation for Top Performing Tools

Verification Tool	FN	FP	TP	TN	FPR	FNR
Symbiotic	2	0	8	3	0	0.2
CBMC	0	1	6	4	0.2	0
CPA-SEQ	2	1	5	3	0.25	0.29

Table 8: Verification Results and Evaluation

Verification Tool	FN	FP	TP	TN	FPR	FNR
Symbiotic	16	3	19	1	0.136	0.45
Pesco	9	6	11	4	0.6	0.45
CBMC	8	4	13	4	0.5	0.38
CpaChecker	16	2	18	1	0.67	0.47
ESBMC-INCR	2	2	4	2	0.5	0.33
CPA-SEQ	8	0	14	2	0	0.36

and security properties and a more comprehensive array of PLC programs.

REFERENCES

[1] Wael Alsabbagh and Peter Langendörfer. 2022. A Flashback on Control Logic Injection Attacks against Programmable Logic Controllers. *Automation* 3, 4 (Nov. 2022), 596–621.

[2] Kaled M Alshmrany, Mohannad Aldughaim, Ahmed Bhayat, and Lucas C Cordeiro. 2021. FuSeBMC: An energy-efficient test generator for finding security vulnerabilities in C programs. In *International Conference On Tests And Proofs*. Springer, 85–105.

[3] Matthias Althoff. 2022. Benchmarks for the Formal Verification of Power Systems. In *Proc. of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems*. mediatum.ub.tum.de.

[4] Jiri Barnat, Lubos Brim, Milan Češka, and Petr Ročkai. 2010. Divine: Parallel distributed model checker. In *2010 ninth international workshop on parallel and distributed methods in verification, and second international workshop on high performance computational systems biology*. IEEE, 4–7.

[5] Dirk Beyer. 2016. Reliable and reproducible competition results with benchexec and witnesses (report on SV-COMP 2016). In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 887–904.

[6] Dirk Beyer. 2022. Cooperative verification: Towards reliable safety-critical systems (invited talk). In *Proceedings of the 8th ACM SIGPLAN International Workshop on Formal Techniques for Safety-Critical Systems*. 1–2.

[7] Dirk Beyer, Jan Haltermann, Thomas Lemberger, and Heike Wehrheim. 2022. Decomposing software verification into off-the-shelf components: an application to CEGAR. In *Proceedings of the 44th International Conference on Software Engineering*. 536–548.

[8] Dirk Beyer and Sudeep Kanav. 2022. CoVeriTeam: On-demand composition of cooperative verification systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 561–579.

[9] Dirk Beyer and M Erkan Keremoglu. 2011. CPAchecker: A tool for configurable software verification. In *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*. Springer, 184–190.

[10] Dirk Beyer, Stefan Löwe, and Philipp Wendler. 2019. Reliable benchmarking: requirements and solutions. *Int. J. Softw. Tools Technol. Trans.* 21, 1 (Feb. 2019), 1–29.

[11] Dirk Beyer and Heike Wehrheim. 2020. Verification artifacts in cooperative verification: Survey and unifying component framework. In *International Symposium on Leveraging Applications of Formal Methods*. Springer, 143–167.

[12] Dimitri Bohlender, Daniel Hamm, and Stefan Kowalewski. 2018. Cycle-bounded model checking of PLC software via dynamic large-block encoding. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing* (Pau, France) *(SAC '18)*. Association for Computing Machinery, New York, NY, USA, 1891–1898.

[13] Lei Bu, Zhunyi Xie, Lecheng Lyu, Yichao Li, Xiao Guo, Jianhua Zhao, and Xuandong Li. 2022. BRICK: Path Enumeration Based Bounded Reachability Checking of C Program (Competition Contribution). In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 408–412.

[14] Matteo Camilli. 2014. Formal verification problems in a big data world: towards a mighty synergy. In *Companion Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE Companion 2014)*. Association for Computing Machinery, New York, NY, USA, 638–641.

[15] Dániel Darvas, Enrique Blanco Vinuela, and Borja Fernández Adiego. 2015. PLCverif: A tool to verify PLC programs based on model checking techniques. (2015).

[16] Xavier Devroey, Alessio Gambi, Juan Pablo Galeotti, René Just, Fitsum Kifetew, Annibale Panichella, and Sebastiano Panichella. 2022. JUGE: An infrastructure for benchmarking Java unit test generators. *Softw. Test. Verif. Reliab.* (Dec. 2022).

[17] Mikhail R Gadelha, Felipe Monteiro, Lucas Cordeiro, and Denis Nicole. 2019. ESBMC v6. 0: Verifying C Programs Using k-Induction and Invariant Inference: (Competition Contribution). In *Tools and Algorithms for the Construction and Analysis of Systems: 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings, Part III 25*. Springer, 209–213.

[18] Mikhail R Gadelha, Felipe R Monteiro, Jeremy Morse, Lucas C Cordeiro, Bernd Fischer, and Denis A Nicole. 2018. ESBMC 5.0: an industrial-strength C model checker. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 888–891.

[19] Fei He, Zhihang Sun, and Hongyu Fan. 2022. Deagle: An SMT-based verifier for multi-threaded programs (competition contribution). In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 424–428.

[20] Eunkyoung Jee, Seungjae Jeon, Sungdeok Cha, Kwangyong Koh, Junbeom Yoo, Geeyong Park, and Poonghyun Seong. 2010. FBDVerifier: Interactive and visual analysis of counter-example in formal verification of function block diagram. *Journal of Research and Practice in Information Technology* 42, 3 (2010), 171–188.

[21] Daniel Kroening and Michael Tautschnig. 2014. CBMC–C Bounded Model Checker: (Competition Contribution). In *Tools and Algorithms for the Construction and Analysis of Systems: 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings 20*. Springer, 389–391.

[22] Reza M. Parizi, Kai Qian, Hossain Shahriar, Fan Wu, and Lixin Tao. 2018. Benchmark Requirements for Assessing Software Security Vulnerability Testing Tools. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 01. 825–826.

[23] Hoda Mehrpouyan, Dimitra Giannakopoulou, Guillaume Brat, Irem Y Tumer, and Chris Hoyle. 2016. Complex engineered systems design verification based on assume-guarantee reasoning. *Systems Engineering* 19, 6 (2016), 461–476.

[24] Hoda Mehrpouyan, Dimitra Giannakopoulou, Irem Y Tumer, Chris Hoyle, and Guillaume Brat. 2014. Combination of compositional verification and model checking for safety assessment of complex engineered systems. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. 46292. American Society of Mechanical Engineers, V01BT02A021.

[25] Hoda Mehrpouyan, Irem Y Tumer, Chris Hoyle, Dimitra Giannakopoulou, and Guillaume Brat. 2014. Formal verification of complex systems based on sysml functional requirements. In *Annual Conference of the PHM Society*, Vol. 6.

[26] Cedric Richter and Heike Wehrheim. 2019. PeSCo: Predicting Sequential Combinations of Verifiers: (Competition Contribution). In *Tools and Algorithms for the Construction and Analysis of Systems: 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings, Part III 25*. Springer, 229–233.

[27] Ruimin Sun, Alejandro Mera, Long Lu, and David Choffnes. 2020. SoK: Attacks on Industrial Control Logic and Formal Verification-Based Defenses. (June 2020). arXiv:2006.04806 [cs.CR]

[28] Haroon Wardak, Sami Zhioua, and Ahmad Almulhem. 2016. PLC access control: a security analysis. In *2016 World Congress on Industrial Control Systems Security (WCICSS)*. ieeexplore.ieee.org, 1–6.

[29] Liangze Yin, Wei Dong, Wanwei Liu, Yunchou Li, and Ji Wang. 2018. YOGAR-CBMC: CBMC with Scheduling Constraint Based Abstraction Refinement: (Competition Contribution). In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 422–426.