
# On-the-fly conformance testing of safety PLC code using QuickCheck



Citation for the original published paper (version of record):
Khan, A., Thonnessen, D., Fabian, M. (2019). On-the-fly conformance testing of safety PLC code using QuickCheck. IEEE International Conference on Industrial Informatics (INDIN), 2019-July: 419-424. http://dx.doi.org/10.1109/INDIN41052.2019.8972277

# On-the-fly conformance testing of safety PLC code using QuickCheck

Adnan Khan*, David Thönnessen†, Martin Fabian*
* Department of Electrical Engineering
Chalmers University of Technology, Göteborg, Sweden
Email: {adnan.khan, fabian}@chalmers.se
† Informatik 11 – Embedded Software
RWTH Aachen University, Aachen, Germany
Email: thoennessen@embedded.rwth-aachen.de

***Abstract*—In this paper, an approach based on the IOCOS testing relation to test safety PLC code using the tool *QuickCheck* is presented. Testing and validation of the safety PLC code is typically carried out on a physical system using checklists. These checklists are developed by engineers using system specification. However, due to the manual nature of checklist generation and execution, certain test cases can be overlooked and can lead to human accidents. The presented approach allows on-the-fly generation and execution of test cases, which expands the scope of testing by including test cases unconceived during checklist generation. Furthermore, it is demonstrated how the errors in the safety PLC code are uncovered based on the IOCOS testing relation.**

***Index Terms*—Safety, PLC, Input-Output Conformance, Model-Based Testing, Discrete Event Systems**

## I. Introduction

Manufacturing and production systems typically consist of many automated machines and robots, which operate in a coordinated manner to achieve a desired task. This coordination is governed by PLCs (Programmable Logic Controllers), which are prone to errors due to manual programming practices. The PLC code can be classified into two categories, one is related to the nominal aspects of the manufacturing system, e.g. on/off operation of a motor, etc. The second category is related to the safety aspects of the system, such as emergency stop.

In an industrial setting, the nominal PLC code and the safety PLC code are typically implemented on separate PLCs. The nominal code is implemented on a standard PLC, while the safety code is implemented on a *safety PLC*. The differences between the two types of PLCs in terms of programming are negligible, but there are some special functions and features in the safety PLC, which are provided to accommodate different safety devices.

After the implementation of the PLC code, testing is carried out to check if the control code conforms to the specification or not. To test the nominal code, a concept called virtual commissioning [1]–[5] is currently being used by many companies. With virtual commissioning, testing of the nominal code is carried out by testing code on a simulation model of the physical system.

The safety code, on the other hand, is typically tested on the physical system using checklists, which are manually generated from verbal descriptions of test cases based on natural language. Thus, these can be interpreted in a variety of ways. Furthermore, the tests are performed manually on the physical system, and the validation is carried out visually. This procedure of visual validation, depending on the size and complexity of the system, can take days and sometimes weeks [6]. Also, the test cases taken into account in the checklists depend on the experience of the tester, any unconceived testing scenario may pose a threat to human life.

The safety logic implemented for manufacturing systems takes control of a physical system in the case of an event, which is typically discrete in nature, e.g. breaking a light curtain, etc. This discrete event driven behavior of the safety logic can be modeled as *discrete event systems* [7]. Discrete event systems evolve with respect to the occurring events and occupy at each instant of time a particular state where certain conditions are valid. There are several formal approaches [7] to implement, analyze, and test discrete event systems.

Model-based testing [8] is one of the formal approaches. It tests a model of an implementation by subjecting it to a series of tests to find errors in it with respect to a given specification. These tests are based on specifications that the implementation should conform to in order to pass the executed test. If the implementation fails the executed test, and upon inspection the specification is found to be correct then the implementation needs to be amended. The model-based testing techniques can be broadly classified into two categories:

- Offline model-based testing
- Online model-based testing

In offline testing, the test cases are generated beforehand (offline) and are executed manually on the implementation. While in online-testing, which is also known as on-the-fly testing, the test cases are generated based on the result of previously executed tests. The assessment of the tests, i.e. if the executed test passed or failed, is decided by an *oracle*. The oracle is an abstract entity that is considered to contain all the

This work has been carried out at the Wingquist Laboratory VINN Excellence Centre within the Production Area of Advance at Chalmers. It has been supported by ITEA3 Vinnova ENTOC (ref 2016-02716), and VR SyTeC (ref 2016-06204).

information necessary to determine whether the implementation conforms or not to the specification.

A formalized version of model-based testing called *input-output conformance* (IOCO) testing was proposed by [9]. For IOCO, the model of an implementation is scrutinized with respect to a specification. If the implementation fails to conform to the specified behaviour then it requires amendment. In the IOCO testing relation, the pass or fail criteria is assessed based on the outputs emitted by the implementation. If other than the specified outputs are emitted, then the implementation is not IOCO with respect to the specification. The conformance relation using IOCO is established solely by checking the emitted outputs and does not have any requirement on the inputs.

The *input-output conformance simulation relation* (IOCOS) proposed by [10] is a stronger relation than IOCO. In addition to IOCO, the IOCOS relation also requires that the implementation accepts a superset of the inputs specified by the specification.

QuickCheck [11] is a Haskell based tool, where system properties are described by Haskell functions, and testing is carried out automatically using user-defined test case generators. QuickCheck has a distinctive attribute called *shrinking*, which in the case of failed executed test cases reduces the initial set of test cases to find the reason of non-conformance of the implementation with respect to the specification. The literature does not, to the best of the author's knowledge, include on-the-fly conformance testing of safety PLC code based on IOCOS using QuickCheck.

### A. Contribution

This paper describes an on-the-fly conformance testing approach for safety PLC code. The presented approach is based on the input-output conformance simulation relation (IOCOS) using the tool QuickCheck. In the proposed approach, QuickCheck initiates test cases by triggering sequences of randomized events in the safety PLC code. After the initiation of the test cases in the safety PLC code, the related inputs and outputs are compared to the specified inputs and outputs to uncover faults based on the IOCOS relation. The oracle is created using the specification in Structured Text and is running on the same PLC, which will transmit the results of the executed tests back to QuickCheck.

### B. Outline

This paper is structured as follows. In Section II, a brief overview regarding current industrial practice of safety PLC code testing is given. Section III introduces the IOCO testing relation and the IOCOS simulation relation in the context of model-based testing. In Section IV, an overview of the proposed approach is detailed, with an implementation example. Section V concludes the paper with future work directions.

## II. Industrial Safety Logic Testing

When a project is conceived, initially its requirement specification is finalized. These specifications contain details regarding equipment (electrical and mechanical) planned to be installed. In addition to the equipment details, the safety aspects are also taken into consideration while finalizing the requirements.

After finalizing the requirement specification, the project is typically outsourced to a contractor. It is now the contractor's responsibility to correctly build the system according to the agreed specification. In current industrial settings, project outsourcing is a common practice among manufacturing companies. The outsourcing on the one hand reduces workload from the company's staff, and on the other hand helps in achieving the desired goal at a reduced cost. But as the product is not being developed in house, it puts more stringent requirements on safety testing with respect to the specification.

The final product testing is typically carried out at the contractor's site. Commonly, an engineer from the manufacturing company with a sound experience in safety testing is assigned to perform this task. This procedure of testing is called *factory acceptance testing* [12]. During the factory acceptance testing phase, the assigned engineer spends several days at the contractor's facility to confirm that the product conforms with the agreed specification.

This conformance relation is tested using manually generated checklists, which are basically series of manual steps applied to trigger the safety logic. These steps typically include pressing certain buttons, e.g. emergency stops, breaking light curtains, etc., while the nominal operation is ongoing. After triggering these events, the engineer observes the behaviour of the manufacturing system and then compares it with the behaviour specified in the checklist. If the exhibited behaviour of the system conforms to the specified behaviour then the executed test is a pass, otherwise it is not.

The scope of the current practice to test the safety code is limited to the checklists. Due to the manual nature of the checklist generation, it is probable that the engineer might not be able to conceive all possible scenarios to test. Hence, some low probability high-risk scenarios not covered in the checklist may still pose a danger to human life.

## III. Conformance Testing

A conformance testing approach proposed by [9] is based on an input-output conformance (IOCO) relation. In this approach, the implementation is tested by executing all possible traces in a specification. After the execution of each trace on the implementation, the outputs emitted by the implementation are compared with the outputs emitted by the specification. The implementation is said to be IOCO only if the implementation emits a subset of the specified outputs. If the implementation emits other than the specified outputs then it is non-IOCO and it has to be amended.

To give the formal definition of IOCO, consider two disjoint sets of input actions $I$ and output actions $O$. The output actions are the actions initiated by the system under test and are expressed with an exclamation mark, such as $!a \in O$. The input actions are commands to the system and are expressed with a question mark such as $a? \in I$. Now, we consider

a labelled transition system in this section to elaborate the concept of IOCO and give the formal definition.

*Definition 1:* An *I/O labelled transition system* comprising inputs and outputs is a 4-tuple $\langle S, s_0, L, \rightarrow \rangle$ where:

- $S$ is a non-empty set of states;
- $s_0 \in S$ is the initial state;
- $L$ is a countable set of labels. These represent observable actions of a system i.e. $L = I \cup O$ where $I$ and $O$ are as above. Consider also a *quiescence* symbol $\delta \notin L$, and define the sets $L_\delta = L \cup \{\delta\}$ and $O_\delta = O \cup \{\delta\}$;
- $\rightarrow \subseteq S \times L_\delta \times S$ is a transition relation such that, $p \xrightarrow{a} q$ implies $\langle p, a, q \rangle \in \rightarrow$ and $p \xrightarrow{a}$ for $a \in L_\delta$, if there exists $q \in S$ such that $p \xrightarrow{a} q$. Similarly, $p \not\xrightarrow{a}$, for $a \in L_\delta$, if there exist no $q$ such that $p \xrightarrow{a} q$. In addition, only coherent quiescent systems are allowed so $\rightarrow$ should also satisfy the following:
  - if $p \xrightarrow{\delta} p'$, then $p = p'$ i.e. a quiescent transition is always reflexive.
  - if $p \not\xrightarrow{!o}$ for all $!o \in O$, then $p \xrightarrow{\delta} p$, i.e. a state with no outputs is quiescent.
  - if $p \xrightarrow{!o}$ for some $!o \in O$, then $p \not\xrightarrow{\delta}$, i.e. a state with some output is not quiescent.

Furthermore, a trace $t$ is a finite sequence of symbols of $L_\delta$ i.e. $t \in L_\delta^*$, including the empty trace $\epsilon$. When the the transition relation is restricted to be a function, and thus for $p \xrightarrow{a} q$ and $p \xrightarrow{a} q'$ it holds that $q = q'$, the resulting LTS is said to be *deterministic*.

Additional definitions needed to express the IOCO relation in Definition 6 are as follows.

*Definition 2:* The set of traces from a state $p$ in an LTS is

$$\textbf{traces}(p) = \{t \in L_\delta^* \mid p \xrightarrow{t}\}. \tag{1}$$

For an LTS $A = \langle S, s_0, L, \rightarrow \rangle$, its set of traces are the ones defined from its initial state

$$\textbf{traces}(A) = \textbf{traces}(s_0). \tag{2}$$

*Definition 3:* The set of states reached *after* a trace $t$ from a state $p$ is

$$\textbf{after}(p, t) = \{p' \in S \mid p \xrightarrow{t} p'\}. \tag{3}$$

For an LTS $A = \langle S, s_0, L, \rightarrow \rangle$, the set of states reached *after* a trace $t$ is

$$\textbf{after}(A, t) = \{p' \in S \mid s_0 \xrightarrow{t} p'\}. \tag{4}$$

For a deterministic LTS, $\textbf{after}(\cdot, \cdot)$ always returns a singleton set. Then we write $\textbf{after}(p, t) = p'$.

*Definition 4:* The set of *outputs* from a state $p$ is

$$\textbf{outs}(p) = \{!x \in O_\delta \mid p \xrightarrow{!x}\}. \tag{5}$$

*Definition 5:* The set of *inputs* for a state $p$ is

$$\textbf{ins}(p) = \{x? \in I \mid p \xrightarrow{x?}\}. \tag{6}$$

The formal definition of the IOCO testing relation [10] can now be stated.

*Definition 6:* For two deterministic LTSs G and S with equal sets of labels, G is said to be IOCO with respect to S if

$$\forall t \in \textbf{traces}(S) : \textbf{outs}(\textbf{after}(G, t)) \subseteq \textbf{outs}(\textbf{after}(S, t)) \tag{7}$$

The formal IOCO definition (Def. 6) is interpreted as an implementation $G$ conforms to a specification $S$, if for all the traces in the specification the outputs possible from the state reached by the implementation after a trace form a subset of the possible output events from the state reached by the specification after the same trace. Whenever this subset relation between the respective sets of output events exist, the implementation is said to be IOCO with respect to the specification, for that particular trace. If the implementation is IOCO with respect to the specification for all the traces defined by the specification, then the implementation is said to be IOCO with respect to the whole specification.

In this paper, we have taken the modified definition of IOCO given by [10], which relaxes the original assumption of the implementation being *input enabled* [9]. In addition, the original version of IOCO considers *suspension traces*, which are the traces containing quiescent behavior (states without any output). But the modified definition takes all the traces into account, because the quiescent behavior is included in the modified definition by introducing a special symbol for it.

According to the formal definition of IOCO, as long as the implementation emits a subset of the specified outputs, it is considered IOCO. But the authors of [10] pointed out a shortcoming in this conformance relation, i.e. the implementation can be IOCO even if it is empty or partially implemented; meaning that some inputs are not implemented but are specified in the specification.

To counter these issues, the authors of [10] proposed a stronger relation called *input-output conformance simulation* (IOCOS). The IOCOS relation puts, in addition to IOCO, a requirement on the implementation that it must conform to at least one of the specified input behaviours, as formally defined in Def. 7. In addition to the IOCO requirements on the implementation having a subset of the specified outputs, IOCOS requires the implementation also to have a super-set of the specified inputs.

The formal definition of the IOCOS simulation relation can now be given.

*Definition 7:* For two deterministic LTSs G and S with equal sets of labels, G is said to be IOCOS with respect to S if in addition to (7), it also holds that

$$\forall t \in \textbf{traces}(S) : \textbf{ins}(\textbf{after}(S, t)) \subseteq \textbf{ins}(\textbf{after}(G, t)) \tag{8}$$

The formal IOCOS definition (Def. 7) states that the implementation $G$ conforms to a specification $S$, if for all the traces in the specification the inputs possible from the state reached by the implementation after a trace form a super-set of the possible input events from the state reached by the specification after the same trace. If this super-set relation between the respective sets of inputs and the subset relation expressed in definition (Def. 6) exist, the implementation is IOCOS with respect to the specification for the executed

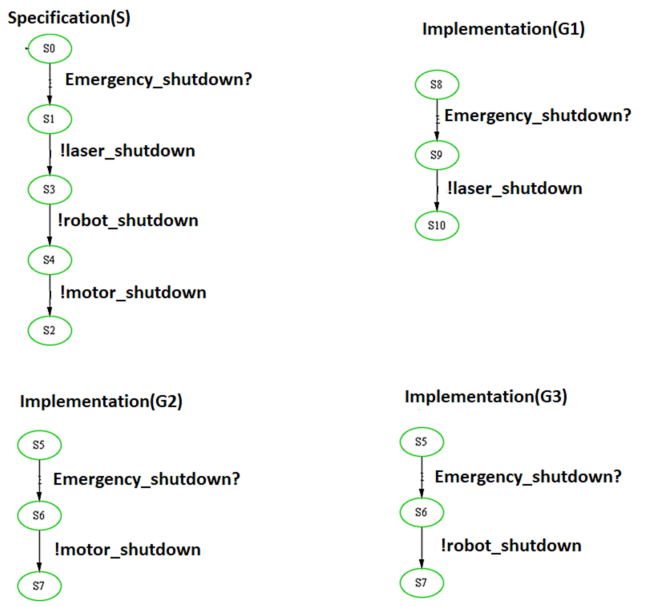


Fig. 1. Emergency shutdown specification and modular implementations

trace. If the implementation is IOCOS with respect to the specification for all the traces defined by the specification, then the implementation is said to be IOCOS with respect to the whole specification.

The IOCOS definition (Def. 7) can be compared to the input-output interpretation of the supervisory control theory proposed by [13], where the supervisor and the plant interact in a symmetric loop so that outputs from the plant are inputs to the supervisor, and outputs from the supervisor are inputs to the plant. The input-output interpretation requires the plant inputs from the supervisor to be accepted by the plant, thus the inputs of the plant should be a super-set of the outputs from the supervisor. Similarly, the outputs generated by the plant are required to be accepted by the supervisor so that the generated outputs of the plant are a subset of the inputs of the supervisor. This latter property, which coincides with IOCO, is called *controllability* [13], [14]. The authors of [15] on the other hand, proposed the idea of a plant as an event generator (both inputs and outputs). According to [15], one can only control the plant by enabling or disabling controllable events (inputs). [13] re-interprets the original theory proposed by [15], where they considered the input events as commands given to the system to which only output events occurs.

## IV. Use Case

For the proposed approach, the testing is carried out on safety PLC code of a real plant. From the perspective of conformance testing, the safety code is the implementation that undergoes testing with respect to the specification. The safety code is created using function block diagrams, while the oracle is implemented based on the specification in Structured Text in the same PLC. After the execution of each test case generated by QuickCheck, the PLC sends the test result, which simply means pass or fail, back to QuickCheck. Based on the test results received, QuickCheck generates new test cases on-the-fly, which are executed on the safety code.

In the proposed use case, the safety code and the specification do not share events, but Boolean signal values. The subset relation between the events is interpreted with Boolean signal values as *False* is a subset of both *False* and *True*, while *True* is a subset of only *True*. Hence, if the specified output is *False* and the implementation output is *True* then the conformance relation fails as the implementation output is not a subset of the specification output.

### A. Safety PLC code

The safety code is implemented in a modular manner for each safety critical event as shown in Fig.1 for each machine in the plant. There are three machines operating in the plant:

- Laser (G1)
- Motor (G2)
- Robot (G3)

These machines remain operational under normal circumstances and their operation is only affected if a safety critical event occur. For the given use case, the safety critical events are:

- Emergency shut-down
- Opening the back-door ports
- Opening the front-door ports
- Opening the operator-door

The emergency shut-down event is activated via six different input signals. Five of these input signals are push-buttons located at different locations within the plant, while one signal comes from the robot control system. According to the safety specification, if the emergency shutdown is triggered via any of these six signals, the nominal operation should be put on halt i.e. the robot, laser, and the motor shall be powered off and require reset to resume nominal operation. Among the three machines, the robot and the laser should stop instantly in case of emergency stop, while the motor shall be stopped 100 milliseconds later.

The safety logic associated to the back-door ports is triggered via two signals. When any of these two signal ports are opened or if the signals coming at the ports fail, the safety logic should deactivate all three machines. Compared to the emergency shutdown logic, the back-door logic deactivates the laser and the robot with a delay of 300 milliseconds, while the motor is stopped after a delay of 600 milliseconds.

For the front-door ports, the safety logic gets triggered via two signals in the same manner as for the back-door ports. The safety code, in case of this event should stop the robot after 300 milliseconds and the remaining two machines should get deactivated after 600 milliseconds. The laser and the motor can remain operational during maintenance as long as the input signal *keyswitch* is true.

Finally, opening of the operator door should affect only the robot via two signals similar to the back-door and front

door ports. The robot in this case should be stopped because the human operator needs to supply some raw material to it. During this situation, the laser and the motor should remain operational.

### B. Test Case Generator

The generator presented in the following serves to generate test cases, more details regarding the implementation are available in [16]. It uses a software testing library called QuickCheck to first come up with randomized test cases and then help to simplify failed test cases in order to support the tester with debugging [11]. The generator requires some elementary information about the system to test. This information consists of two parts and has to be defined by the tester.

The first part is a set of events $E$ triggering one sensor of the system each. It is focused on sensors that supervise the system for hazardous behavior or giving an operator the ability to shut down the system in case of an emergency. Triggering is further partitioned into so-called *negative* and *positive* events, given as $E_n$ and $E_p$ with $E = E_n \cup E_p$. Negative events are the set of events that are meant to shut down the system or parts of it, for example pressing a specific emergency button. Positive events are the set of events with inverted semantics, they allow the system to recover. For every negative event $n_i \in E_n$ there is a positive event $p_i \in E_p$. There are functions $p : E_n \to E_p$ and $n : E_p \to E_n$ mapping events of each set to the other. When referring to either a positive or negative event the notation $e_i \in E$ is used. The corresponding positive event for the previously given example of pressing an emergency button might be to release the button. Nevertheless, the tester has to define this plant-specific behavior. It has to be noted that a positive event is not required to set the system into operation again, this is implementation-specific and does not have to be taken into account by the test case generator.

The second part consists of a range of durations of events. The range $D_R := [d_{\text{min}}, d_{\text{max}}]$ is given by the tester and the generator is free to choose a duration $d_i \in D_R$ for each event. Every event $e_i$, positive or negative, is combined with a duration $d_i \in D_R$. A duration can be understood as a pause between two events. It describes the amount of time reserved for executing an event. Another event cannot be executed before the duration of the preceding event has elapsed. A duration of zero milliseconds has the semantics of two events happening at the same time. To allow the generator to come up with reasonable event durations, the tester has to specify the desired range of durations, given by the minimum and maximum duration. In order to keep the time consumption of executing test cases low, it is recommended to not choose high maximum durations. The tester has to ensure that no system behavior is masked by too low maximum durations.

Using these inputs, the generator returns a test case consisting of an event sequence $S := \{(e_1, d_1), (e_2, d_2), \ldots, (e_n, d_n)\}$ with $n \in \mathbb{N}_{\geq 0}$. An event $e_i$ is executed at time $t_i = \sum_{j=0}^{i-1} d_j$, which is the sum of the durations of all preceding events.Events are chosen by picking one of all available events with an evenly distributed probability $p_i = n^{-1}$ with $n$ being the number of events. It is not considered whether a door was already opened or not. For example, closing an already closed door has no effect. The duration of events is estimated by a randomized calculation rule. With $50\,\%$ probability a duration $d_i$ of $d_{\text{min}}$ is chosen. With the remaining $50\,\%$ probability a duration $d_i \in D_R$ is chosen. This choice is done evenly distributed over the whole range.

TABLE I
EXEMPLARY GENERATED EVENT SEQUENCE.

| $i$ | $e_i$ | $d_i$ |
|---|---|---|
| 1 | EMERGENCY_LEFT_PRESS | 0 ms |
| 2 | DOOR_OPERATOR_OPEN | 1886 ms |
| 3 | EMERGENCY_LEFT_RELEASE | 396 ms |
| 4 | DOOR_BACK_OPEN | 0 ms |
| 5 | DOOR_OPERATOR_CLOSE | 170 ms |

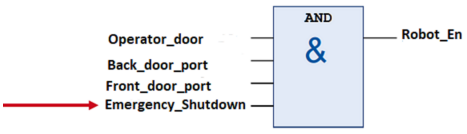


Fig. 2. Missing input event for robot

An example event sequence of five events with a duration range $D_R := [0\,\text{ms}, 2000\,\text{ms}]$ is given in Table I. The given sequence first triggers to press an emergency button 'LEFT'. $0\,\text{ms}$ later, meaning at the same time, the door 'OPERATOR' is opened. The just pressed emergency button is released after further 1886 milliseconds and so on.

In the implementation of the testing environment, an initial event sequence size of 50 events is chosen. After executing a test case, the result is propagated to the tester as *passed* or *failed*. QuickCheck offers a feature called *shrinking* which tries to simplify a failed test case to a less complex one. The presented generator makes use of this and basically works as follows. If a test case passed, another test case is executed as there is no shrinking necessary. If a test case failed, shrinking tries to reduce the number of events by choosing different subsets of the failed event sequence. The choice of subsets is implemented in QuickCheck and not part of this work. If the number of events is greater than one, but cannot be reduced further without having the test case pass, then shrinking tries to omit single events $e_i$ by adding their duration $d_i$ to the preceding event $e_{i-1}$ such that $d'_{i-1} = d_{i-1} + d_i$. If all combinations are exhausted, shrinking tries to reduce the duration of each event, but only one per try, to $d_{\text{min}}$. In case this makes the test case pass again, the duration is gradually increased. The shrinking process is finished when there are no options left to reduce the complexity of the test case without having it pass.

### C. Testing

The tests executed by QuickCheck either pass or fail using the specifications as the basis. For each safety critical input event e.g. $Emergency\_shutdown$, there are associated output events modelled e.g. $motor\_shutdown$, $laser\_shutdown$, $robot\_shutdown$, as shown in Fig.1, which get updated in the safety code. These updated output values, after the execution of each test, are compared with the specified output values. For the safety code to be IOCOS with respect to the specification, each input event in the specification should be implemented in the safety code, and the updated output values of the safety code after the execution of the test must be a subset of the specified values.

All executed tests but one were IOCOS with respect to the specification. The failure occurred when the input event $Emergency\_shutdown$ was triggered. Upon manual inspection, it was found that the safety code fulfills the IOCOS requirement for the outputs as it emitted a subset of the specified outputs. However, the input $Emergency\_shutdown$ was not found in the robot function block of the safety code. Due to this, the IOCOS requirements related to inputs expressed in definition (Def. 7) was not fulfilled. To make the safety code IOCOS, the missing input was added in the safety code, see Fig.2.

During testing a randomized test sequence generated by QuickCheck also caused a failure, which upon inspection was found to be IOCOS. However, the failure occurred due to a discrepancy between the oracle and the safety code. The triggered sequence of events that caused the failure were $Emergency\_shutdown$, $back\_door\_port$, and deactivation of the $Emergency\_shutdown$. In this sequence, when the $Emergency\_shutdown$ event was triggered, the robot and the laser stopped instantly, while the motor stopped with a delay of 100 milliseconds, which is correct safety behaviour per specification for this event. The occurrence of the second event $back\_door\_port$ immediately after $Emergency\_shutdown$ did not affect the status of the machines as they were stopped already. However, when the $Emergency\_shutdown$ event was deactivated in the presence of the $back\_door\_port$ event, all machines became operational for some time. This so, because the timers associated with the $back\_door\_port$ event were not elapsed while the $Emergency\_shutdown$ was deactivated. Consequently, the system state was set to run until the timers elapsed. This failure is due to momentary activation of machines, which could lead to human accident in real industrial setting.

This test case appears to be difficult to conceive manually while preparing a checklist. Also, the IOCOS formalism is found to be not detailed enough to express timed behaviour associated with inputs and outputs. Therefore, in order to test the safety PLC code, the IOCOS relation requires modification.

## V. Conclusion

In this paper, an on-the fly conformance testing approach to test safety PLC code using QuickCheck is presented. The proposed approach is applied on safety PLC code of a real plant to demonstrate the IOCOS relation. The results of the testing highlighted the advantages of using on-the-fly test generation and execution via QuickCheck. Furthermore, the limitations of IOCOS for testing the safety PLC code is highlighted. The IOCOS requires the implementation to have a subset of the outputs and a super-set of the specified inputs, while the IOCO relation only requires the implementation to have a subset of the specified outputs. Therefore, this subset requirement for both inputs and outputs has revealed that neither IOCO nor IOCOS relation is suitable for safety code testing, because for safety each specified input and output must be implemented. Otherwise, due to the subset relation, some safety behaviour can go untested, which can lead to human accidents. In addition, it is shown that the current IOCOS relation is not detailed enough to capture timing discrepancies. In future work, the IOCOS relation will be examined and modified for safety.

## References


[1] Z. Liu, C. Diedrich, and N. Suchold, *Virtual Commissioning of Automated Systems*. INTECH Open Access Publisher, 2012.

[2] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian, and P. Falkman, "Integrated virtual preparation and commissioning: supporting formal methods during automation systems development," *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 1939–1944, 2016.

[3] P. Hoffmann, R. Schumann, T. M. Maksoud, and G. C. Premier, "Virtual commissioning of manufacturing systems a review and new approaches for simplification." in *24th European Conference on Modelling and Simulation (ECMS 2010)*, 2010, pp. 175–181.

[4] A. Jain, D. Vera, and R. Harrison, "Virtual commissioning of modular automation systems," *IFAC Proceedings Volumes*, vol. 43, no. 4, pp. 72–77, 2010.

[5] C. G. Lee and S. C. Park, "Survey on the virtual commissioning of manufacturing systems," *Journal of Computational Design and Engineering*, vol. 1, no. 3, pp. 213–222, 2014.

[6] L. Fransén, *National Electric Vehicle Sweden, Manufacturing Engineer, Private conversation*, 30 March 2016.

[7] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, ser. SpringerLink Engineering. Springer US, 2009.

[8] M. Utting and B. Legeard, *Practical Model-Based Testing: A Tools Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[9] G. Tretmans, "Test generation with inputs, outputs and repetitive quiescence, 1996," *URL http://doc. utwente. nl/65463*, vol. 46, 1996.

[10] C. Gregorio-Rodríguez, L. Llana, and R. Martínez-Torres, "Input-output conformance simulation (iocos) for model based testing," in *Formal Techniques for Distributed Systems*. Springer, 2013, pp. 114–129.

[11] K. Claessen and J. Hughes, "Quickcheck: a lightweight tool for random testing of haskell programs," *Acm sigplan notices*, vol. 46, no. 4, pp. 53–64, 2011.

[12] A. Rudqvist, *GKN aerospace, System Owner Robotics, Simulation and OLP, E-mail conversation*, 09 April 2018.

[13] S. Balemi, "Control of discrete event systems: theory and application," Ph.D. dissertation, Swiss Federal Institute of Technology, Zürich, Switzerland, 1992.

[14] T. Jéron, H. Marchand, V. Rusu, and V. Tschaen, "Ensuring the conformance of reactive discrete-event systems using supervisory control," in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, vol. 3. IEEE, 2003, pp. 2692–2697.

[15] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[16] D. Thönnessen, N. Smallbone, M. Fabian, and S. Kowalewski, "Testing Safety PLCs Using QuickCheck," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2019, handed in.