Enforcing I/O sequences for PLC validation purposes

Anaïs Guignard, Jean-Marc Faure

Enforcing I/O sequences for PLC validation purposes

Anais Guignard
Automated Production Research Laboratory LURPA
Ecole Normale Superieure de Cachan, 61 Avenue du President Wilson, F-94230 Cachan
anais.guignard@lurpa.ens-cachan.fr

Jean-Marc Faure, *Member, IEEE*
Automated Production Research Laboratory LURPA
Ecole Normale Superieure de Cachan, 61 Avenue du President Wilson, F-94230 Cachan
jean-marc.faure@lurpa.ens-cachan.fr

Abstract

Validation of the behavior of a Programmable Logic Controller (PLC) by comparison of observed I/O sequences to sequences built from a formal specification model requires that the consequences of the PLC I/O scanning cycle be considered. This paper proposes a method based on an enforcement technique to interpret observed I/O sequences so that the result of this comparison be meaningful.

1 Introduction

As defined in [3], validation consists in checking that the product does the right thing or, with other words, that the product conforms to its specification. Programmable Logic Controllers (PLC) are products which are more and more integrated in automated systems, even to perform critical functions; this explains why validation of PLC is gaining an always increasing interest.

A possible solution to meet this objective is to apply formal verification techniques ([2]) on the specification of the control logic ([4], [13]) or the PLC code that implements this logic ([1], [5], [6], [9], [12]). These techniques are based on an exhaustive analysis of a state space which represents the specification or the PLC code according to the verification objective. However, they operate on a model and not a real device. Hence they are helpful but not sufficient to validate a real PLC that executes a control code.

Validation of a real PLC may be performed by conformance test ([7], [11]). In this approach, the PLC is not connected to the plant that it must control but to a test-bench; the PLC output sequences in response to input sequences generated by the bench are compared to the expected sequences. A second solution for PLC validation is to connect the controller to the plant, or to a simulation of the plant (Hardware In the Loop approach), to observe the input/output (I/O) sequences in this closed-loop system and to compare them with the expected sequences. The first solution requires a wider state space be analyzed, because the PLC behavior is not constrained by that of the plant, then a longer validation time. The second solution focuses only on the useful functional behavior (only the constrained state space of the PLC behavior is considered) and will be selected in this work.

However, comparison of observed and theoretical expected I/O sequences require the technological features of the PLC, and in particular its I/O scanning cycle, be considered. A negative comparison verdict indeed may come from a flawed control algorithm or code or a difference introduced by this cycle; these two cases must be distinguished. The aim of this paper is to show that an enforcement technique permits to meet this objective. The modeling formalism which has been selected to specify the expected behavior of the PLC is presented in the next section. Some phenomena introduced by the I/O scanning cycle are presented in section 3. Section 4 presents a brief reminder on the selected enforcement technique, enforcement by edit automaton, and the main contribution of this paper, a method to construct from a specification model an enforcement monitor. This method is illustrated in section 5. Concluding remarks and some perspectives are given in the last section.

2 Description of the expected behavior

This section presents the formalism selected to represent the expected behavior of a PLC whose input and output Boolean variable sets are respectively I_{PLC} and O_{PLC}: Mealy machines.

A Mealy machine is a 6-tuple (I_M, O_M, S_M, s_{init}, δ_M, λ_M) where:

- I_M is the input alphabet; $|I_M| = 2^{|I_{PLC}|}$ is the size of this alphabet and each element I_j is a Boolean combination of elements of I_{PLC}

- O_M is the output alphabet; $|O_M| = 2^{|O_{PLC}|}$ is the size of this alphabet and each element O_j is a Boolean combination of elements of O_{PLC}
- S_M is a finite set of states s_i
- s_{init} is the initial state
- $\delta_M : S_M \times I_M \rightarrow S_M$ is the transition function
- $\lambda_M : S_M \times I_M \rightarrow O_M$ is the output function

In the sequel of this paper, the Mealy machine is assumed to be deterministic (δ_M is a function), minimal and completely defined (δ_M is defined for every couple (s_i,i_j)).

An expected sequence σ_{exp} is a sequence of Input/Output (I/O) vectors built by firing a set of successive transitions of the Mealy machine. This sequence corresponds to a conform behavior and will be written:

$$\sigma_{exp} = \left(\begin{pmatrix} I_{PLC} \\ O_{PLC} \end{pmatrix}_1^{exp}, \cdots, \begin{pmatrix} I_{PLC} \\ O_{PLC} \end{pmatrix}_n^{exp} \right) \quad (1)$$

The construction of the edit automaton that will allow a consistent comparison of an observed sequence with an expected sequence will be illustrated in this paper on the example of Figure 1. This Mealy machine describes the expected behavior of a simple logic controller with two input variables $I_{PLC} = \{i_1, i_2\}$ and two output variables $O_{PLC} = \{o_1, o_2\}$; its input and output alphabets are respectively $I_M = \{i_1 \cdot i_2, \overline{i_1} \cdot i_2, i_1 \cdot \overline{i_2}, \overline{i_1} \cdot \overline{i_2}\}$[1] and $O_M = \{o_1 \cdot o_2, \overline{o_1} \cdot o_2, o_1 \cdot \overline{o_2}, \overline{o_1} \cdot \overline{o_2}\}$.

An I/O vector associated to a transition of the machine is written under the form:

$$I/O = \begin{pmatrix} i_1 \\ i_2 \\ o_1 \\ o_2 \end{pmatrix}$$

A possible expected sequence can be built from this example by defining the following scenario of successive input changes:

- When the initial state is the active state, all inputs are False; the two outputs are False too.
- Then, both inputs are simultaneously set to True; the active state becomes the state 2 and both outputs become True.
- The input variable i_2 is reset (becomes False) afterwards; the machine remains in the current state and the outputs are not modified.

[1] $(\cdot)$ represents the operator of disjunction and $(\bar{\ })$ represents the complement

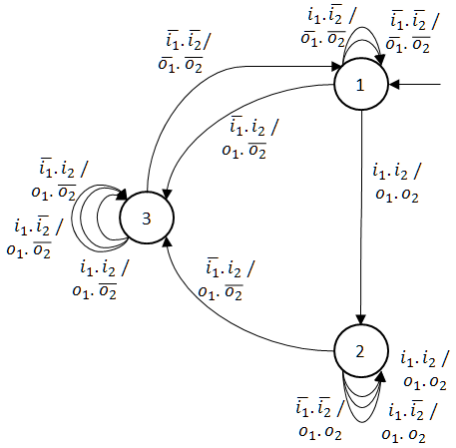

Figure 1. Example of specification

Therefore, the corresponding I/O expected sequence for this scenario, where 1 means True and 0 False, is:

$$\sigma_{exp} = \left(\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} \right) \quad (2)$$

3 Observed I/O sequences features

To validate a PLC in real condition of operation, it has to be connected to a plant, or to a software that simulates the plant (Hardware in the Loop). Then, the values of the inputs and outputs of the controller can be obtained at the output and input interfaces of the plant (an output of the plant is an input of the controller and vice versa); it is assumed that every change of an input/output value is detected and that only of the I/O vectors that differ from the previous one are kept. The experimental input and output variable sets are I_{PLC} and O_{PLC}. Then, an observed sequence σ_{obs} corresponds to the sequence of I/O vectors recorded during HIL simulation.

$$\sigma_{obs} = \left(\begin{pmatrix} I_{PLC} \\ O_{PLC} \end{pmatrix}_1^{obs}, \cdots, \begin{pmatrix} I_{PLC} \\ O_{PLC} \end{pmatrix}_n^{obs} \right) \quad (3)$$

Each observed I/O vector comprises an input vector defined on the set I_{PLC} and an output vector defined on the set O_{PLC}. It must be noted that the definitions of the expected and observed sequences are similar but that the expected sequence contains input/output values obtained from the specification model whereas the observed sequence contains experimental values. Last, it must be underlined that the HIL simulation is not used to validate directly the controller, by mere observation of the plant evolutions by a human operator, but to obtain observed I/O sequences that will be compared later to expected ones.

This work assumes that the PLC which is to be validated is mono-task; this assumption is quite reasonable

because most of controllers that are integrated in automated systems use this kind of task scheduling. In this case, the execution of the control algorithm is based on a cyclic I/O scanning that can be periodic or not. Each I/O scanning cycle comprises 4 phases:

- Inputs reading,
- Internal and output variables computation,
- Outputs updating,
- Waiting time until the end of the period, if the cycle is periodic.

An observed I/O sequence may differ from the expected one because the PLC code includes some flaws (the validation verdict must be then negative) but also because the I/O scanning cycle may provoke three phenomena that are not possible in the formal model: delay on output emission, synchronization of asynchronous input variables or desynchronization of synchronous input variable. Only the last phenomenon will be considered in this paper because even if it is less known and treated than the synchronization phenomenon, its effects are not negligible. Indeed, it is not always possible to build a specification model in the form of a Mealy machine that does not include simultaneous input changes for two successive transitions, as discussed in [10]. Indeed, this phenomenon happens when two input simultaneous changes occur during the reading phase. As this phase is not instantaneous, one input change may be detected and the other one not, as illustrated on Figure 2.

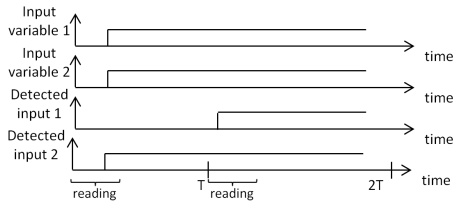

Figure 2. Desynchronization phenomenon of synchronous events during inputs reading[2]

This phenomenon and its potential consequences on the observed I/O sequences are illustrated hereafter on the example of section 2, where two input values are changed from the first to the second vector. It must be underlined that the values of the inputs and outputs are observed from the plant, i. e. outside of the PLC in what follows.

The observed sequence

$$\sigma_{obs1} = \left(\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right) \quad (4)$$

[2]Only two inputs are represented in these timing diagrams but the reasoning can be extended to more than two inputs.

can be explained by a desynchronization such that i_2 is detected by the PLC cycle before i_1. The values of the outputs in the second vector point out that the controller has evolved to a state that corresponds to the state 3 of the specification (o_1 True and o_2 False); the second observed I/O vector does not comply with the specification, because it is not possible to have these values of the outputs when both inputs are True from the initial state.

The observed sequence

$$\sigma_{obs2} = \left(\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} \right) \quad (5)$$

can be explained by a desynchronization such that i_1 is detected by the PLC cycle before i_2. The values of the outputs in the second and third vectors point out that the controller has performed two evolutions which correspond first to a self-loop on the initial state then to a transition to the state 2 of the specification. However, all combinations of inputs and outputs in this sequence comply with the specification.

To distinguish these two cases, the observed sequence must be interpreted before being compared with the expected sequence. This interpretation of the observed sequence according to the possible consequences of the I/O scanning cycle can be performed by using enforcement techniques, as it will be shown in section 4. These techniques have been developed recently as an alternative for software verification during runtime to the well-known model-checking techniques that require a model be built and that often lead to state space explosion when analyzing this model; enforcement techniques operate on sequences of events, then do not require any model and are claimed less sensitive to combinatory explosion. A brief description of the enforcement technique that has been selected in this study is given in the next section before presenting the application to the addressed issue.

4 Enforcement by edit automaton

4.1 Definition

Several enforcement techniques have been proposed, for timed or non-timed systems. The technique based on an edit automaton, introduced in [8], has been selected in this work because it is quite suitable to solve the issue that has been pinpointed in the previous section and can be relatively easily applied when the specification model is a Mealy machine, as it will be shown in the next section. In this approach (Figure 3), an enforcement monitor receives a sequence of events σ that may or not satisfy a given property ϕ (notation $\sigma \models \phi$?) and generates an output sequence o that satisfies this property (notation $o \models \phi$). This monitor is composed of an automaton, termed edit automaton, and a memory to store a sequence of events.

The following notations will be used:

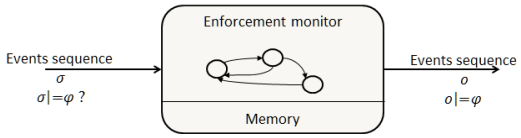

Figure 3. Principle of enforcement by edit automaton

- Σ is the set of observable events.[3] In this study, an event a is an observed I/O vector.
- Σ^* is the set of all finite sequences over Σ. Σ^* represents then the set of all possible observed sequences.
- $\sigma \in \Sigma^*$ is a finite sequence of events, then an observed sequence.

The symbol Ø denotes the empty sequence, $\sigma[i]$ denotes the $i-th$ term in the sequence σ. Let σ and τ be two sequences of actions. $\sigma;\tau$ is the concatenation of both sequences and $\sigma \prec \tau$ (resp. $\sigma \succ \tau$) means that σ is a subsequence of τ (resp. τ is a subsequence of σ).

With these notations, an edit automaton can be defined as a $6-tuple$ $(\Sigma, Q, q_0, \delta, \gamma_o, \gamma_k)$ where:

- Σ is a finite non-empty set of observable events,
- Q is a finite set of states,
- q_0 is the initial state,
- $\delta : Q \times \Sigma \rightarrow Q$ is a labeled partial transition function,
- $\gamma_o : Q \times \Sigma \rightarrow \Sigma^*$ defines the sequence emitted when firing a transition,
- $\gamma_k : Q \times \Sigma \rightarrow \Sigma^*$ defines the sequence kept in memory.

Let $\sigma = a;\sigma'$ be a sequence where a is the first event; four cases are generally possible to define γ_o and γ_k:

- Validation transition (Figure 4.a) with $\gamma_o(q,\sigma) = a$ and $\gamma_k(q,\sigma) = \sigma'$; the event a is emitted and the rest of the sequence is kept.
- Suppression transition (Figure 4.b) with $\gamma_o(q,\sigma) =$ Ø and $\gamma_k(q,\sigma) = \sigma'$; no event is emitted and the rest of the sequence is kept.
- Insertion transition (Figure 4.c) with $\gamma_o(q,\sigma) = a'$ and $\gamma_k(q,\sigma) = \sigma$; a new event a' is emitted and the complete sequence σ is kept.
- Stop transition (Figure 4.d) with $\gamma_o(q,\sigma)$ =Ø and $\gamma_k(q,\sigma) =$ Ø; no event is emitted and the whole sequence is deleted.

[3] In the original paper which introduced the edit automata, this set was termed the 'set of actions'. We have thought that 'set of events' is clearer for readers with a background in Discrete Event Systems theory.

The first case means that the property ϕ is satisfied when a is emitted and the second case that the event a must not be emitted to satisfy ϕ. The third case means that another event must be emitted to satisfy ϕ (a will be emitted or not in a next transition) while the fourth case that it is no more possible to satisfy ϕ; the outgoing sequence is then stopped.

On Figure 4, the label above the transition (a) is the first event of the incoming sequence and the label below the transition is the result of the function γ_o (emitted event).

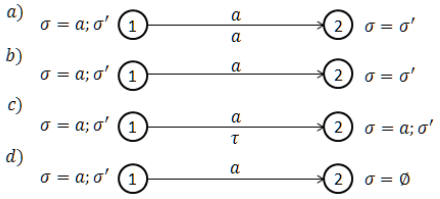

Figure 4. Possible transitions of an edit automaton

4.2 Building an edit automaton from a Mealy machine

The aim of this section is to propose a method to construct, from a Mealy machine which represents the specified behavior of a controller, an edit automaton that will be able to distinguish the two kinds of observed sequences discussed at section 3: sequences where all I/O vectors comply with the specification (relation 5) and sequences that contain at least one I/O vector which does not comply with the specification (relation 4).

Construction of this edit automaton requires that the consequences of the PLC I/O scanning cycle be integrated in the formal model (Figure 5). As only one consequence, desynchronization of synchronous input changes is considered in this paper, the edit automaton will include new states and transitions that correspond to this only phenomenon; the principle of the construction could be applied for the other consequences, however.

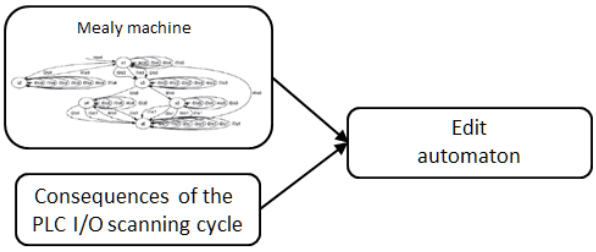

Figure 5. Construction of the edit automaton

The edit automaton is constructed in two steps. The first step is a simple syntactic transformation where a first

structure of the edit automaton is derived from that of the Mealy machine:

- The set of states of the automaton is the same than that of the machine.
- As an event is an I/O vector in this work, the set of events is easily derived from the input and output alphabets of the machine.
- A transition of the automaton is created for each transition of the machine; this transition is a validation transition where the incoming sequence is reduced to the single event constructed from the input and output labels of the corresponding transition of the machine.

In the second step, states that represent the possible I/O vectors which can be observed when desynchronization occurs and comply with the specification are introduced for each transition that may suffer from the desynchronization phenomenon; these new states are named *intermediate states*. Suppression transitions from the specified state to the intermediate states are then added; these transitions model changes of the I/O vector that are not represented in the theoretical specification model but may be observed and will lead to the specified state, target of the specified transition. Last, validation transitions are added from the intermediate states to the downstream state of the considered transition.

At the end of the treatment, the edit automaton contains two types of states: specified states, obtained at the end of the first step, and intermediate states, added during the second step, and two kinds of transitions: specified transitions which are derived directly from the specification model and new suppression or validation transitions that represent possible changes of the observed I/O vector that are allowed because they can lead to a specified state. An observed I/O sequence that will be declared conform to the specification is a path between two specified states; this path may cross intermediate states or not depending on the occurrence of desynchronization.

5 Illustration

The construction method described in the previous section will be illustrated on a reduced part of the example of Figure 1: the transition from the state 1 to the state 2. If this transition is fired immediately after the self-loop on the state 1 where both inputs are False, two simultaneous changes of inputs will occur in the theoretical model but these changes may be desynchronized by the PLC scanning cycle. The method must be applied obviously to the similar transitions starting from the other states.

Once this part of the edit automaton will be obtained, it will be shown how the two observed I/O sequences given at the end of section 3 (definitions 4 and 5) are enforced by this automaton.

5.1 Construction of the edit automaton

The result of the first step of the construction method is shown on Figure 6. The input/output labels of the Mealy machine are replaced by I/O vectors and all transitions are validation transitions, i.e. when the enforcement monitor receives the corresponding I/O vector, this vector is emitted and nothing is stored in the memory of the monitor.

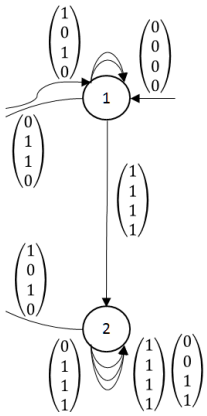

Figure 6. Result of the first step

The result of the second step is presented in Figure 7. The intermediate state 4 corresponds to an I/O vector where no output has changed and the intermediate states 5 and 6 to an I/O vector where respectively only o_1 or o_2 has changed. The transitions from the state 1 to these intermediate states and between two intermediate states are suppression transitions; the I/O vector is represented with pale characters in this case. The transitions from the intermediate states to the state 2 are validation transitions; the I/O event is represented with dark characters.

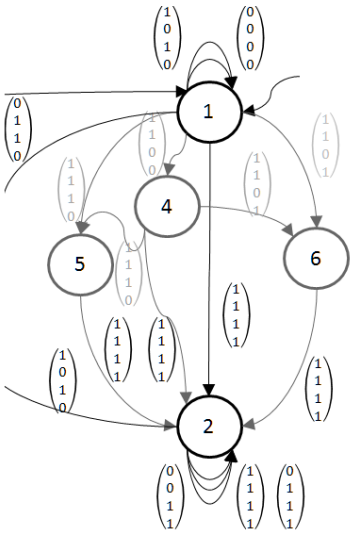

Figure 7. Result of the second step

It must be underlined that this automaton remains deterministic because the events associated to the new transitions starting from the state 1 are different from those of the initial transitions, derived directly from the Mealy ma-

chine, because they correspond to I/O combinations which are not possible in the specification.

5.2 Enforcing observed I/O sequences

When the other observed I/O sequence (4) is applied to this automaton, the second I/O vector provokes the suppression transition to the state 6; this vector is then suppressed. Moreover, no transition is starting from this state for the third I/O vector; this means that the enforcement of the sequence must be stopped. Implicit stop transitions start indeed from every state for all I/O vectors that are not associated to existing validation, suppression or insertion transitions starting from this state. The resulting enforced sequence is:

$$\sigma_{enforced1} = \left(\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \right) \quad (6)$$

and points out that the evolution was not that expected.

When the observed I/O sequence (5) is applied to this automaton, from its initial state, the first I/O vector, which corresponds to a self-loop on the initial state, is emitted. The second I/O vector provokes the suppression transition to the state 4; this vector is therefore suppressed. Then, the third I/O vector provokes the validation transition to the state 4 and is emitted; this is also the case for the fourth I/O vector which causes a self-loop validation transition on the state 2. Finally, the enforced sequence is:

$$\sigma_{enforced2} = \left(\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} \right) \quad (7)$$

and is identical to that expected. The event consequence of the PLC scanning cycle (second I/O vector) has been suppressed by enforcement.

6 Conclusion

This paper has shown how an enforcement technique, enforcement by an edit automaton, may be employed to interpret an I/O sequence obtained by observation of the evolutions of a closed loop system. This technique allows in particular an observed sequence be compared to an expected sequence, built from a specification model, even if these two sequences include different numbers of I/O vectors because the PLC I/O scanning cycle has introduced new vectors in the observed sequence.

However, this work has considered only one consequence of the I/O scanning cycle: desynchronization of theoretically synchronous events. The next step is to extend the approach to the other consequences. This might lead however to increase significantly the number of new states and transitions. To prevent from this immoderate growth of the state space, enforcement approaches based on the representation of the property to satisfy by rules and not by an automaton are under investigation.

References

[1] H. Bel Mokadem, B. Berard, V. Gourcuff, O. De Smet, and J.-M. Roussel. Verification of a timed multitask system with UPPAAL. *Automation Science and Engineering, IEEE Transactions on*, 7(4):921–932, 2010.

[2] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[3] B. W. Boehm. *Classics in software engineering*. Yourdon Press, Upper Saddle River, NJ, USA, 1979.

[4] G. Frey and L. Litz. Formal methods in PLC programming. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 4, pages 2431–2436 vol.4, 2000.

[5] V. Gourcuff, O. De Smet, and J.-M. Faure. Improving large-sized PLC programs verification using abstractions. In *Proceedings of the 17th IFAC World Congress*, pages 5101–5106, July 2008.

[6] S. Lamperiere-Couffin and J.-J. Lesage. Formal verification of the sequential part of PLC programs. In R. Boel and G. Stremersch, editors, *Discrete Event Systems*, volume 569 of *The Springer International Series in Engineering and Computer Science*, pages 247–254. Springer US, 2000.

[7] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8):1090–1123, Aug. 1996.

[8] J. Ligatti, L. Bauer, and D. Walker. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 4:2–16, 2005.

[9] O. Pavlovic and H.-D. Ehrich. Model checking plc software written in function block diagram. In *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, pages 439–448, 2010.

[10] J. Provost, J.-M. Roussel, and J.-M. Faure. SIC-testability of sequential logic controllers. In *10th International Workshop on Discrete Event Systems*, pages 203–208, 2010.

[11] J. Provost, J.-M. Roussel, and J.-M. Faure. Translating Grafcet specifications into Mealy machines for conformance test purposes. *Control Engineering Practice*, 19(9):947–957, Sept. 2011.

[12] B. Schlich, J. Brauer, and S. Kowalewski. Direct model checking of plc programs in IL. In *Dependable Control of Discrete Systems*, volume 2, pages 28–33, 2009.

[13] V. Vyatkin and H.-M. Hanisch. A modeling approach for verification of IEC1499 function blocks using net condition/event systems. In *Emerging Technologies and Factory Automation, 1999. Proceedings. ETFA '99. 1999 7th IEEE International Conference on*, volume 1, pages 261–270 vol.1, 1999.