Validation of logic controllers from events observation in a closed-loop system

Anaïs Guignard, Jean-Marc Faure

Validation of logic controllers from events observation in a closed-loop system

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

Numerous worthwhile results have been published in the last two decades on validation of logic controllers by using formal methods like model-checking or conformance testing. Whatever the merits of these contributions, the first approach considers only a model of the control code while the second one focuses on an isolated controller that executes this code. However, from a control engineering point of view, validation of a logic controller requires also to analyze the behavior of the controller when it is connected to the plant it must control to form a closed-loop system. This paper proposes a method to check, from observation of I/O events, whether the behavior of such a controller conforms to its specification. The principle of this method is to build a model of the closed-loop system from the observed I/O events then to compare this model to the specification model. A criterion to terminate the observation step is defined by using previous results on identification of discrete event systems. This method is illustrated on a small example.

1 Introduction

As defined in [2], 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. This paper focuses on the validation of a mono-tasking PLC with cyclic input and output (I/O) scanning that executes an implementation of a logic control that has been specified to control a given plant.

A possible solution to meet this objective is to use formal verification techniques, like model-checking [4][1]. This valuable approach ensures an exhaustive analysis of a model of the logic control but not of an implementation of this control. Therefore, it is surely necessary but not sufficient to fully validate a PLC.

A promising approach to analyze an implemented control is conformance testing [3][8][10]. This approach can be decomposed in two phases: test sequence construction and test execution. The aim of the first phase is to build an input sequence, called test sequence, from the specification model. A test objective must be defined before building this sequence; for non-timed critical systems whose specification is described by a finite state machine, an usual test objective is to cross at least once every transition of the machine. This input sequence is then applied to the implementation under test, a PLC in this paper, during test execution. The sequence of outputs that are emitted by the PLC is then observed and compared with the expected output sequence defined by the specification.

It must be clearly underlined that, during the execution of the test, the PLC is connected to a test-bench that generates the input sequence and observes the output sequence but not to the plant which must be controlled. Hence, the observed behavior is that of the PLC in isolation and not of the PLC in a closed-loop. The state space that corresponds to the observations is therefore larger than the state space that describes the evolutions of the PLC connected to the plant, because in the latter case the evolutions of the PLC are constrained by those of the plant. This implies that the results of the conformance test cannot be completely applied in the real operational conditions. In particular, a negative conformance verdict may come either from an error in the implemented logic control or from a situation that is not possible in the closed-loop because it is not reachable owing to the structure of the plant (limitations on some movements, inter-locked actuators, etc.).

This explains why we propose in this paper a method to validate a PLC that executes an implementation of a logic control from observation of I/O events in a closed-loop system. It will be assumed that the plant is faultless and that no plant model is available; only a model of the specification of the logic control is given. Validation will be performed by observing the I/O events of the controller, building a model of the closed-loop system from these observations then comparing this model to the specification model, as illustrated at Figure 1. It may be noted that the closed-loop includes a real PLC and a real or simulated plant; the last solution fits very well the recent needs for HIL (Hardware In the Loop) validation where a model of the plant is connected to a real controller.

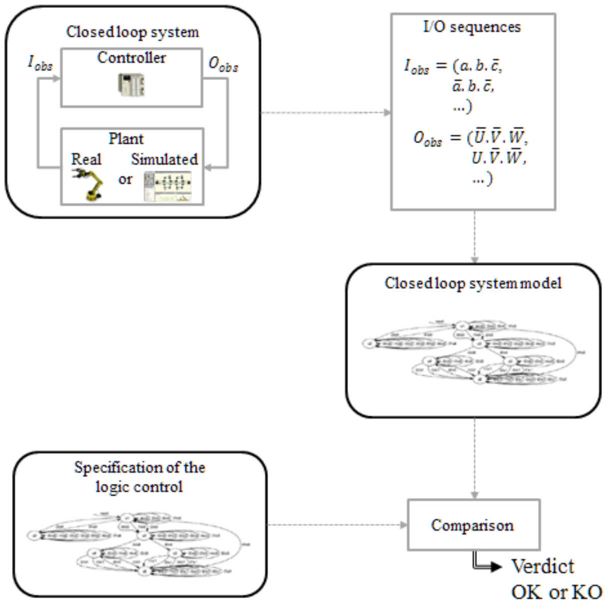
Closed loop system
I_{obs}
Controller
O_{obs}
Plant
Real
Simulated
or
I/O sequences
$I_{obs} = (a.b.\bar{c}, \bar{a}.b.\bar{c}, ...)$
$O_{obs} = (\bar{U}.\bar{V}.\bar{W}, U.\bar{V}.\bar{W}, ...)$
Closed loop system model
Specification of the logic control
Comparison
Verdict OK or KO

Figure 1. Objective of the work

The main issue in this approach is to define a criterion to terminate the observation step. A too short observation will provide indeed an incomplete model of the closed-loop system which will not be appropriate for comparison with the specification model; a too long observation will surely hinder the applicability of the approach. To solve this issue, a termination criterion that has been previously introduced for identification of discrete event systems has been selected.

After two reminders on the formalism selected to specify the logic control and on identification of discrete event systems in section 2, the third section provides a global description of the validation method we propose. The second step of this method, construction of the model of the closed-loop system, is formally defined at section 4. A complete example is developed in section 5 while some conclusions and perspectives are drawn up in section 6.

2 Background

2.1 Specification model

In this paper, the specification model is assumed to be a Mealy machine. This choice has been motivated by the fact that a Mealy machine is a formal class of finite automata for which transitions are labeled by two elements that represent the input values read and the output values emitted by the controller in one cycle.

Formally, the specification model is a Mealy machine defined by a 6-tuple (I_{Spec}, O_{Spec}, S_{Spec}, s_{init}, δ_{Spec}, λ_{Spec}) where:

- I_{Spec} is the input alphabet
- O_{Spec} is the output alphabet
- S_{Spec} is a finite set of states s_i
- s_{init} is the initial state
- $\delta_{Spec} : S_{Spec} \times I_{Spec} \to S_{Spec}$ is the transition function
- $\lambda_{Spec} : S_{Spec} \times I_{Spec} \to O_{Spec}$ is the output function

Let V_I be the set of input variables and V_O be the set of output variables of the PLC. The input alphabet I_{Spec} (resp. output alphabet O_{Spec}) of the machine is composed of all combinations of input (output) variables. The dimension of I_{Spec} (O_{Spec}) is $|I_{Spec}| = 2^{|V_I|}$ ($|O_{Spec}| = 2^{|V_O|}$) and each element of the input (output) alphabet is represented by a minterm[1] defined on V_I (V_O). Hence, every input (output) event of the model corresponds to a combination of the input (output) variables of the PLC

The following assumptions are to be made to build a conformance test sequence from this specification model:

- The transition function is complete and deterministic (i.e. every transition $\delta(s \times i_{Spec})$ with $s \in S$ and $i_{Spec} \in I_{Spec}$ is defined once and only once).
- Each state is distinguishable from the other ones by observation of the output.
- There is no transient evolution, i.e. no input change causes successive changes of state or emitted output.

A simple Mealy machine where these assumptions are verified is given at Figure 2.

2.2 Identification of discrete event systems

Identification of a discrete event system (DES) consists in building a formal model of this system from observation of I/O events and possibly from a priori knowledge about its internal structure. The formal model can be represented by using finite automata [7][11] or Petri Nets [9][6]. Identification techniques may be classified

[1]A minterm defined on a set of n Boolean variables is the conjunction of all these variables in their positive or complemented form.

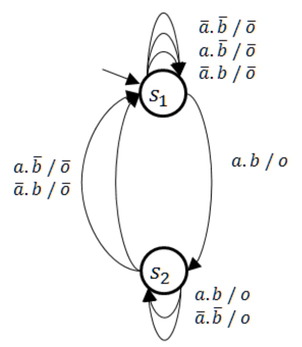

Figure 2. A basic example of Mealy machine with distinguishable states and no transient evolution

into two categories: black-box identification and gray-box identification. No knowledge on the system structure is available in the first approach while partial information about the system to identify is given in the second one. [5], for instance, presents a gray-box identification technique for Petri Nets by assuming that either the number of places and transitions or an upper bound of the number of places is known.

This work considers only identification of DES represented by automata because the specification model is given in the form of an automaton; as the principle of the validation method which is proposed is to compare a model constructed from observations with the specification model, the choice of the formalism is straightforward. Moreover, the construction of the model of the closed-loop system is an example of gray-box identification where the specification model is a superset of the model to build, as it will be detailed in section 3.

The issue mentioned at the end of the introduction: definition of an appropriate termination criterion of the observation step, will be solved by using the results presented in [7]. This reference describes an identification technique to build an automaton that will be used later as a reference model for diagnosis. The identified model must generate every observed event sequence of given length, and only these sequences; a termination criterion of the observation step to meet this objective is defined.

3 Overall description of the approach

The aim of this work is to propose a method to validate a PLC that executes an implementation of a logic control from observation of I/O events in a closed-loop system. This method comprises three steps:

- Observation of I/O events,
- Construction of the model of the closed-loop system from these observations,
- Comparison of this model with the specification model.

The last two steps can be performed off-line, once a sequence of events is observed, or on-line, as soon as a new I/O event is observed.

3.1 Observation of I/O events

An input (respectively output) event corresponds to the change of at least one input (output) variable of the PLC. An I/O event is then built by reading the input and output variables (information from/to the plant) at the end of each PLC cycle and keeping only the combinations which correspond to the change of at least one variable. It will be assumed in what follows that every change is detected.

A sequence of I/O events is merely built by concatenation of I/O events.

3.2 Construction of the model of the closed-loop system

This model is a priori unknown and must be built by identification. A gray-box approach can be used, because the language accepted by the model of the closed-loop system is included inside the language of the specification, if the controller conforms to the specification, as it is detailed below.

3.2.1 Controller, plant and closed-loop system models

If the specification is modeled by a Mealy machine, the plant and the controller must obviously be modeled by Mealy machines which share the alphabets I_{Spec} and O_{Spec}; I_{Spec} (O_{Spec}) is the input (output) alphabet of the controller model and the output (input) alphabet of the plant model. The two models are therefore synchronized and the evolutions of each of them are restricted by the other model; a plant model may, for instance, represent constraints on sensors values (i.e. two limit switches are never True at the same time in a faultless plant) that limit the evolutions of the controller model.

The set of all evolutions of an automaton can be represented by a language. If L_p is the language accepted by the model of the plant and L_c the language accepted by the model of the controller[2], the language accepted by the model of the closed-loop system L_{cls} is the intersection of these two languages, as represented at Figure 3. This figure shows that the language L_{cls} is included into the specification language. Hence, identification of the closed-loop system consists in finding a subset of the specification model that complies with the observed I//O events, assuming that the controller conforms to the specification.

3.2.2 Construction method

From the previous analysis, each observed I/O event may be interpreted as the label of a transition from an upstream state to a downstream state in the specification

[2]If the controller conforms to its specification, this model is the specification model.

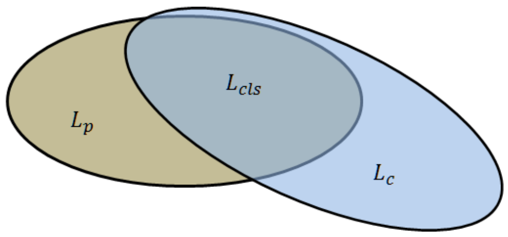

Figure 3. Language of the closed-loop system

model. Therefore, the principle to construct the model of the closed-loop, starting from the initial state s_{init}, is to select the states and transitions in the specification model that correspond to the observed I/O events. The algorithm that performs this operation is formally defined at section 4.

3.2.3 Termination of the observation step

If the model of the closed-loop system was identical to the specification model, the observation step would be finished when the firing of every transition of the specification was observed at least once. This condition does not commonly hold however. The model to identify is not complete and the firing of some transitions will be never observed. To determine the termination of the observation step, the following strategy has been developed (Figure 4):

- Each time the firing of a new transition is observed, a counter is incremented; the current value of this counter represents the number of transitions whose at least one firing has been observed.
- When the value of this counter has not changed during a pre-defined time, the observation step stops. The criterion to terminate the observation step is thus that no firing of a transition which has not been previously fired has been observed during a sufficiently long time.

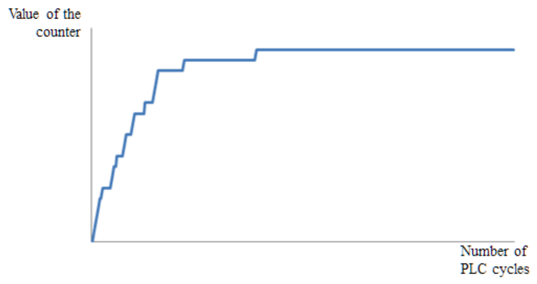

Figure 4. Evolution of the number of observed fired transitions

A similar strategy has been already proposed in [7] for black-box identification of a closed-loop system. In this case, no a priori knowledge about the model to identify was available and it was not possible to reason with the transitions of an already known model. Hence, the event sequences of a given length that were not already observed were counted; it is shown experimentally in this reference that the number of these sequences converges to an upper limit. The observation is stopped when this limit is reached.

3.3 Validation of the controller

The discussion of 3.2 assumed that the controller conformed to the specification. If this is not the case, for at least one state of the specification model, at least one observed I/O event will not correspond to the label of a transition starting from this state. Hence, if it is possible for the whole sequence of observed I/O events, to interpret every event as the firing of a transition of the specification model that starts from the active state, the controller is declared valid; otherwise, it does not conform to the specification. With other words, validation can be performed during the construction of the model of the closed-loop system. If this model can be fully identified from the specification model and the whole sequence of observed I/O events, the controller is validated.

4 Construction of the model of the closed-loop system

This section will present the algorithm to build the model of the closed-loop system and validate the controller. This model is a Mealy machine defined by a 6-tuple (I_{cls}, O_{cls}, S_{cls}, $s_{initcls}$, δ_{cls}, λ_{cls}). According to the previous section, its two alphabets are identical to those of the specification. Its initial state is also the initial state of the specification; the implementation of the logic control in the PLC would be non-conform otherwise. The set of states S_{cls}, as well as the transition and output functions are subsets of the corresponding elements of the specification, however. The previous 6-tuple is therefore formally defined as follows:

- $I_{cls} = I_{Spec}$
- $O_{cls} = O_{Spec}$
- $S_{cls} \subseteq S_{Spec}$
- $s_{initcls} = s_{init}$
- $\delta_{cls} \subseteq \delta_{Spec}$
- $\lambda_{cls} \subseteq \lambda_{Spec}$

The construction of this model from a sequence of observed I/O events is described by the Algorithm 1. Only the initial state is known at the beginning and there is no element in the transition function δ_{cls} and in the output function λ_{cls}. For each observed I/O event, the algorithm verifies whether there exists a transition labeled by this event in the specification model. If this transition exists and if no firing of this transition has been observed before, the algorithm creates the corresponding transition and destination state in the model of the closed-loop system (lines 4 to 12). The analysis continues from this state. When the observed I/O event cannot be interpreted as the firing of a transition of the specification, the construction is stopped and the controller is not validated (line 14).

We can notice that this algorithm can be performed online. This means that the construction is done and the verdict on validity is given at each observation step and the process can be stopped as soon as a non expected output is observed.

Algorithm 1 Construction of the model of the closed-loop system

```
1:  Let $\sigma = (IO_1, ..., IO_n)$ be the observed sequence
    with $IO_k = (i_k, o_k)$ where $i_k \in I_{Spec}$ and $o_k \in O_{Spec}$
2:  Let $s = s_{initcls}$ be the current state of the identified
    model; $S_{cls} = s_{initcls}$
3:  for $IO = (i, o) \in \sigma$ do
4:      if it exists $s_d \in S_{Spec}$ such as $\delta_{Spec}(s, i) = s_d$
    and $\lambda(s, i) = o$ then
5:          if $s_d \notin S_{cls}$ then
6:              $S_{cls} = S_{cls} \cup s_d$
7:          end if
8:          if no firing of the transition $\delta_{cls}(s, i)$ has been
    previously observed then
9:              $\delta_{cls}(s, i) = s_d$
10:             $\lambda_{cls}(s, i) = o$
11:         end if
12:         $s = s_d$
13:     else
14:         The controller does not conform to the speci-
    fication
15:     end if
16: end for
```

5 Example

5.1 Specification model

The basic example presented Figure 5[3][4] will be used to illustrate the results of this work. This model represents the specification of a logic controller with four input variables $V_I = \{G_o, G_c, Car, Rec\}$ and two output variables $V_O = \{Open, Close\}$. The input and output alphabets of this Mealy machine comprise respectively 16 and 4 Boolean combinations of variables. The machine includes 3 states and 48 transitions (16 transitions for every state to ensure completeness).

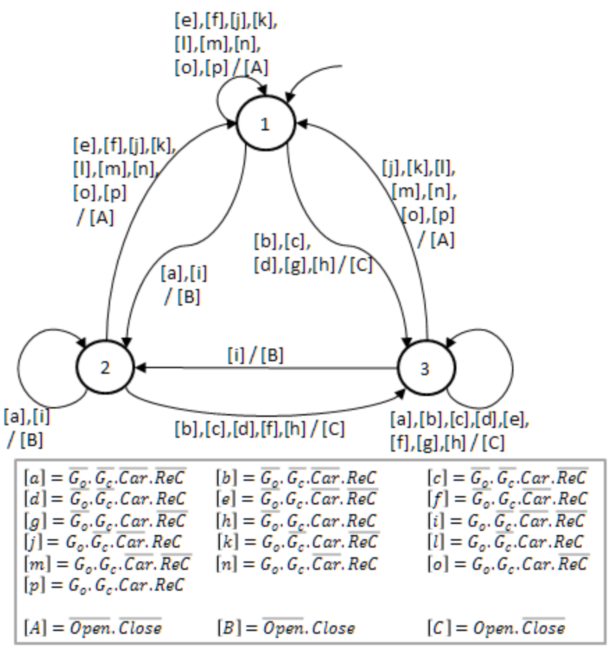
[e],[f],[j],[k], [l],[m],[n], [o],[p] / [A]
1
[e],[f],[j],[k], [l],[m],[n], [o],[p] / [A]
[j],[k],[l], [m],[n], [o],[p] / [A]
[a],[i] / [B]
[b],[c], [d],[g],[h] / [C]
[i] / [B]
2
3
[a],[i] / [B]
[b],[c],[d],[f],[h] / [C]
[a],[b],[c],[d],[e], [f],[g],[h] / [C]
$[a] = \overline{G_o}.\overline{G_c}.\overline{Car}.\overline{ReC}$
$[b] = \overline{G_o}.\overline{G_c}.\overline{Car}.ReC$
$[c] = \overline{G_o}.\overline{G_c}.Car.\overline{ReC}$
$[d] = \overline{G_o}.\overline{G_c}.Car.ReC$
$[e] = \overline{G_o}.G_c.\overline{Car}.\overline{ReC}$
$[f] = \overline{G_o}.G_c.\overline{Car}.ReC$
$[g] = \overline{G_o}.G_c.Car.\overline{ReC}$
$[h] = \overline{G_o}.G_c.Car.ReC$
$[i] = G_o.\overline{G_c}.\overline{Car}.\overline{ReC}$
$[j] = G_o.\overline{G_c}.\overline{Car}.ReC$
$[k] = G_o.\overline{G_c}.Car.\overline{ReC}$
$[l] = G_o.\overline{G_c}.Car.ReC$
$[m] = G_o.G_c.\overline{Car}.\overline{ReC}$
$[n] = G_o.G_c.\overline{Car}.ReC$
$[o] = G_o.G_c.Car.\overline{ReC}$
$[p] = G_o.G_c.Car.ReC$
$[A] = \overline{Open}.\overline{Close}$
$[B] = \overline{Open}.Close$
$[C] = Open.\overline{Close}$

Figure 5. Specification model

5.2 Example of observed I/O sequence

The input and output values are observed in the PLC and are stored under the form :

$$IO = \left(\begin{pmatrix} G_o \\ G_c \\ Car \\ ReC \end{pmatrix}, \begin{pmatrix} Open \\ Close \end{pmatrix} \right) \tag{1}$$

The beginning of the observed sequence is[5]:

$$\sigma_{obs} = \left(\left(\begin{pmatrix} F \\ T \\ F \\ F \end{pmatrix}, \begin{pmatrix} F \\ F \end{pmatrix} \right), \left(\begin{pmatrix} F \\ T \\ T \\ F \end{pmatrix}, \begin{pmatrix} T \\ F \end{pmatrix} \right), \left(\begin{pmatrix} F \\ F \\ T \\ F \end{pmatrix}, \begin{pmatrix} T \\ F \end{pmatrix} \right), \left(\begin{pmatrix} F \\ F \\ F \\ F \end{pmatrix}, \begin{pmatrix} T \\ F \end{pmatrix} \right), \left(\begin{pmatrix} T \\ F \\ F \\ F \end{pmatrix}, \begin{pmatrix} F \\ T \end{pmatrix} \right), \left(\begin{pmatrix} F \\ F \\ F \\ F \end{pmatrix}, \begin{pmatrix} F \\ T \end{pmatrix} \right), \left(\begin{pmatrix} F \\ T \\ F \\ F \end{pmatrix}, \begin{pmatrix} F \\ F \end{pmatrix} \right), \ldots \right) \tag{2}$$

[3]The operator $(\cdot)$ represents the operator of conjunction and $(\bar{\ })$ represents the complement

[4]For clarity reasons, when several transitions start from the same upstream state and arrive to the same downstream state, only an arrow is drawn between these states.

[5]Where T (resp. F) means that the value of the variable is $True$ (resp. $False$)

5.3 Construction of the model of the closed-loop system

As the initial state is known and the specification model is deterministic, it is thus possible to find, for each I/O event, the only corresponding transition and its downstream state, if the controller conforms to the specification. For example, the first observed I/O event is:

$$IO = \left(\begin{pmatrix} F \\ T \\ F \\ F \end{pmatrix}, \begin{pmatrix} F \\ F \end{pmatrix} \right)$$

with $i_{Spec} = [e]$ and $o_{Spec} = [A]$. From the initial state (state 1) of the specification model, this I/O event corresponds to the self-loop on this state labeled $[e]/[A]$. The transition and output functions are respectively $\delta_{Spec}(1,[e]) = 1$ and $\lambda_{Spec}(1,[e]) = [A]$.

If a fired transition in the specification model is written:

$$t_{fired} = \begin{pmatrix} s_u \\ s_d \\ i_{Spec} \end{pmatrix}$$

where s_u, s_d and i_{Spec} represent respectively the upstream state, downstream state and input event of this transition, the sequence of fired transitions in the specification model that corresponds to (2) is:

$$\begin{aligned} T_{fired} = \Bigg(& \begin{pmatrix} 1 \\ 1 \\ \text{[e]} \end{pmatrix}, \begin{pmatrix} 1 \\ 3 \\ \text{[g]} \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \\ \text{[g]} \end{pmatrix}, \\ & \begin{pmatrix} 3 \\ 3 \\ \text{[c]} \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \\ \text{[a]} \end{pmatrix}, \begin{pmatrix} 3 \\ 2 \\ \text{[i]} \end{pmatrix}, \\ & \begin{pmatrix} 2 \\ 2 \\ \text{[i]} \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \\ \text{[a]} \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \\ \text{[e]} \end{pmatrix}, \\ & \begin{pmatrix} 1 \\ 1 \\ \text{[e]} \end{pmatrix}, \ldots \Bigg) \end{aligned} \tag{3}$$

It must be noted that ten transition firings are associated to only seven observed events. The second event corresponds indeed to the firing of a transition that provokes a change of the active state (from state 1 to state 2) and is immediately followed by the firing of the self-loop on the new active state.

The whole identified model of the closed-loop system is presented at Figure 6. This model contains 3 states and 36 transitions. This means that 12 transitions of the specification model that correspond to the combinations of input values where G_c and G_o are both $True$, are not possible in the model of the closed-loop.

5.4 Termination of the observation

The evolution of the counter of observed fired transitions for the complete I/O sequence is represented at Figure 7. The value of this counter converges to 36; no firing of a transition which has not been fired previously is observed once this value reached.

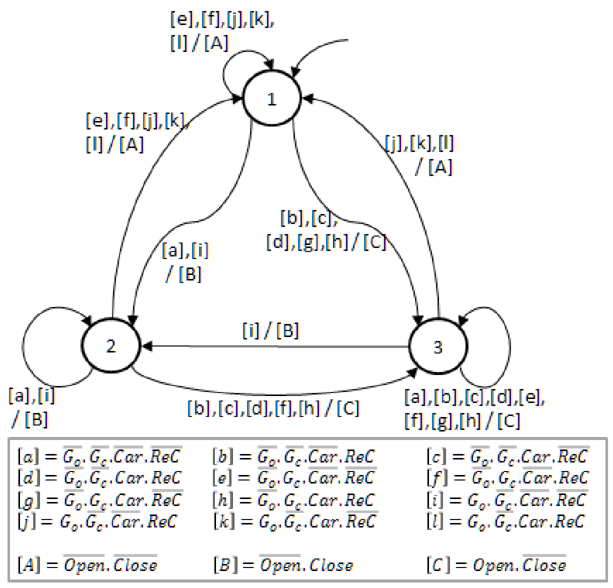
[e],[f],[j],[k],
[l] / [A]
1
[e],[f],[j],[k],
[l] / [A]
[j],[k],[l]
/ [A]
[b],[c],
[d],[g],[h] / [C]
[a],[i]
/ [B]
[i] / [B]
2
3
[a],[i]
/ [B]
[b],[c],[d],[f],[h] / [C]
[a],[b],[c],[d],[e],
[f],[g],[h] / [C]
$[a] = \overline{G_o}.\overline{G_c}.\overline{Car}.\overline{ReC}$
$[b] = \overline{G_o}.\overline{G_c}.\overline{Car}.ReC$
$[c] = \overline{G_o}.\overline{G_c}.Car.\overline{ReC}$
$[d] = \overline{G_o}.\overline{G_c}.Car.ReC$
$[e] = \overline{G_o}.G_c.\overline{Car}.\overline{ReC}$
$[f] = \overline{G_o}.G_c.\overline{Car}.ReC$
$[g] = \overline{G_o}.G_c.Car.\overline{ReC}$
$[h] = \overline{G_o}.G_c.Car.ReC$
$[i] = G_o.\overline{G_c}.\overline{Car}.\overline{ReC}$
$[j] = G_o.\overline{G_c}.\overline{Car}.ReC$
$[k] = G_o.\overline{G_c}.Car.\overline{ReC}$
$[l] = G_o.\overline{G_c}.Car.ReC$
$[A] = \overline{Open}.\overline{Close}$
$[B] = \overline{Open}.Close$
$[C] = Open.\overline{Close}$

Figure 6. Identified model of the closed-loop system

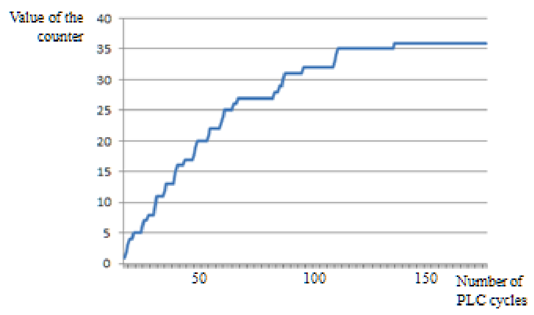

Figure 7. Number of observed fired transitions for the complete I/O sequence

5.5 Case of a non valid controller

It has been assumed in 5.3 and 5.4 that the controller conformed to the specification. The aim of this section is to show how a flawed implementation of the specification can be detected during the construction of the model of the closed-loop system. It will be assumed to meet this objective that the following I/O event sequence is observed from the initial state:

$$\sigma_{obs} = \left(\left(\begin{pmatrix} F \\ T \\ F \\ F \end{pmatrix}, \begin{pmatrix} F \\ F \end{pmatrix} \right), \left(\begin{pmatrix} F \\ T \\ T \\ F \end{pmatrix}, \begin{pmatrix} T \\ F \end{pmatrix} \right), \left(\begin{pmatrix} F \\ F \\ T \\ F \end{pmatrix}, \begin{pmatrix} T \\ F \end{pmatrix} \right), \left(\begin{pmatrix} F \\ F \\ F \\ F \end{pmatrix}, \begin{pmatrix} F \\ F \end{pmatrix} \right) \right) \quad (4)$$

The first three I/O events are identical to those of (2); only the fourth one is different. Hence, by using the Algorithm 1, the first four transitions of the sequence of fired transitions derived from these observations (5) are identical to those of (3). The fifth one is the empty set, however. When the fourth event is observed indeed, the state 3 is the active state but there is no transition with the corresponding label ([a]/[A]) that starts from this state in the model of Figure 5; the controller does not conform to the specification.

$$T_{fired} = \left(\begin{pmatrix} 1 \\ 1 \\ \text{[e]} \end{pmatrix}, \begin{pmatrix} 1 \\ 3 \\ \text{[g]} \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \\ \text{[g]} \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \\ \text{[c]} \end{pmatrix}, \emptyset \right) \quad (5)$$

6 Conclusion and perspectives

This paper has presented a method to validate a PLC from observation of I/O events in a closed-loop system. The main interest of this work is to provide a verdict about the correctness of the controller from observations in real operational conditions and not in specific test conditions. The principle of this contribution is to build a model of the closed-loop system by checking whether the complete sequence of observed I/O events can be interpreted as a sequence of transitions of the specification model; if this is not the case, the PLC is not validated. The whole method has been also illustrated on a simple but relevant example.

On-going work is aiming at extending these results by considering that the I/O events are observed outside the PLC, while it has been assumed in this paper that they are obtained from the internal variables of the PLC. This new approach will require to integrate in the analysis the delays and unexpected synchronization mechanisms introduced by the I/O scanning cycle. In spite of these difficulties, this new approach seems promising for HIL systems where the PLC is connected to a simulated plant because it does not require any knowledge of the internal structure of the controller.

7 Acknowledgment

This work is funded by the French research agency (ANR) as part of the VACSIM project (ANR-11-INSE-004).

References

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

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

[3] T. Chow. Testing software design modeled by finite-state machines. *Software Engineering, IEEE Transactions on*, SE-4(3):178–187, May 1978.

[4] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

[5] M. Dotoli, M. P. Fanti, and A. M. Mangini. Real time identification of discrete event systems using Petri nets. *Automatica*, 44(5):1209–1219, May 2008.

[6] A.-P. Estrada-Vargas, E. Lopez-Mellado, and J.-J. Lesage. Automated Modelling of Reactive Discrete Event Systems from External Behavioural Data. In *Proc. of CONIELECOMP'13*, pages pp. 120–125, Cholula Puebla, Mexique, Mar. 2013.

[7] S. Klein, L. Litz, and J.-J. Lesage. Fault detection of Discrete Event Systems using an identification approach. In *16th IFAC world Congress*, Praha, Czech Republic, July 2005.

[8] 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.

[9] M. Meda-Campana, A. Ramirez-Treviro, and E. Lopez-Mellado. Asymptotic identification of discrete event systems. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, volume 3, pages 2266–2271 vol.3, 2000.

[10] J. Provost, J.-M. Roussel, and J.-M. Faure. Testing Programmable Logic Controllers from Finite State Machines specification. In *3rd International Workshop on Dependable Control of Discrete Systems - DCDS 2011*, pages 0–0, Saarbrücken, Allemagne, June 2011. 6 pages.

[11] M. Roth, J. Lesage, and L. Litz. Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems. In *American Control Conference (ACC)*, pages 2601–2606, 2010.