HAL
open science

Generation of Single Input Change Test Sequences for Conformance Test of Programmable Logic Controllers

Julien Provost, Jean-Marc Roussel, Jean-Marc Faure

To cite this version:

Julien Provost, Jean-Marc Roussel, Jean-Marc Faure. Generation of Single Input Change Test Sequences for Conformance Test of Programmable Logic Controllers. IEEE Transactions on Industrial Informatics, 2014, pp.1. 10.1109/TII.2014.2315972 . hal-00982396

HAL Id: hal-00982396

https://hal.science/hal-00982396v1

Submitted on 23 Apr 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Generation of Single Input Change Test Sequences for Conformance Test of Programmable Logic Controllers

Julien Provost, *Member, IEEE*, Jean-Marc Roussel, Jean-Marc Faure, *Member, IEEE*

***Abstract*—Conformance test is a functional test technique which is aiming to check whether an implementation, seen as a black-box with inputs/outputs, conforms to its specification. Numerous theoretical worthwhile results have been obtained in the domain of conformance test of finite state machines. The optimization criterion which is usually selected to build the test sequence is the minimum-length criterion. Based on experimental results, this paper focuses on the generation of a Single Input Change (SIC) test sequence from a specification model represented as a Mealy machine; such a sequence is aiming at preventing from erroneous test verdicts due to incorrect detection of synchronous input changes by the Programmable Logic Controller (PLC) under test. A method based on symbolic calculus to obtain the part of the specification that can be tested with a SIC sequence is first presented. Then, an algorithm to build the SIC test sequence is detailed; three solutions are proposed, according to the connectivity properties of the SIC-testable part.**

***Index Terms*—Conformance test, Formal Methods, Programmable Logic Controller, Test Sequence, Mealy machine, Test Verdict, Single Input Change**

I. Introduction

PROGRAMMABLE LOGIC CONTROLLERS (PLCs) are industrial automation components that are widely used to implement control functions, even in critical systems like power production and distribution, rail transport, chemical processes, water distribution, etc. This explains why numerous research works have been carried out since more than ten years to develop methods that avoid flaws to be introduced during the development of PLC software [1]. These researches are based on two main approaches: model-based (model-driven) engineering [2]–[5] and formal verification and validation (V&V) techniques [6]–[9] or a combination of both [10]. Whatever the interest of the results obtained in these works, it must be noted that all of them are based on *models*. Formal V&V techniques for instance have been applied to models of the specification of the control logic, in the form of IEC 60848 models, state-charts, Signal Interpreted Petri Nets, Net Condition Event Systems [11]–[13] or of PLC programs, in IEC 61131-3 or IEC 61499 languages [14]–[19].

However, validation of a *real* PLC which executes a control program requires the conformance test of this component be performed, even if the specification and program models have been verified and validated, and a certified code generator has been used to produce the executable code. Conformance test is a functional test, i.e. the system under test, named implementation, is seen as a black-box (its internal structure is unknown) with observable inputs/outputs; the overall aim is to check whether this implementation behaves as specified (see Fig. 1). Conformance test of PLCs is advocated by certification bodies and standards [20], [21], which explains the growing interest of companies in several industrial domains for efficient hardware-in-the-loop techniques [22]–[24] to improve the existing practices.

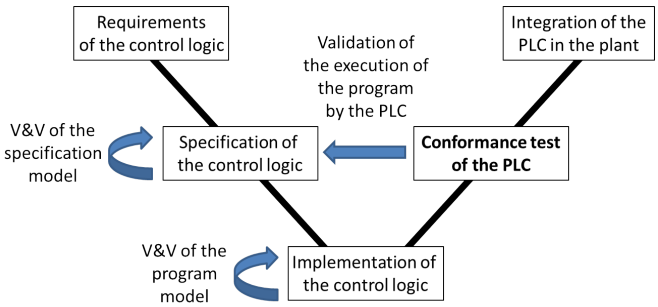

Fig. 1. Place of the work in the life-cycle of PLC software

A promising solution to develop such techniques is to benefit from the results of the researches of the Discrete Event Systems community in the domain of conformance test of formal models. In these works, the specification is a formal model: a Mealy (or finite state) machine [25], a transition system [26], [27], or a timed automaton [28] for instance. The first formalism has been selected for this study because it is well suited to the modeling of logic systems specifications; moreover, functional correctness must be tested before time correctness. As industrial specifications are not expressed in formal languages but in standardized, tailored-made languages, translation rules of industrial specification languages into formal ones are to be developed in order to use these theoretical results for conformance test of PLCs; this issue has been solved in [29] where a method to obtain

Manuscript received February 5 2014; accepted March 28, 2014

J. Provost is with the Assistant Professorship for Safe Embedded Systems, Technische Universität München, DE-85748 Garching bei München, Germany. E-mail: julien.provost@tum.de

J.-M. Roussel and J.-M. Faure are with LURPA, ENS Cachan, FR-94230 Cachan, France. E-mail: jean-marc.roussel@lurpa.ens-cachan.fr, jean-marc.faure@lurpa.ens-cachan.fr

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

© 2014 IEEE - Author version. Citation information: DOI 10.1109/TII.2014.2315972, IEEE Transactions on Industrial Informatics

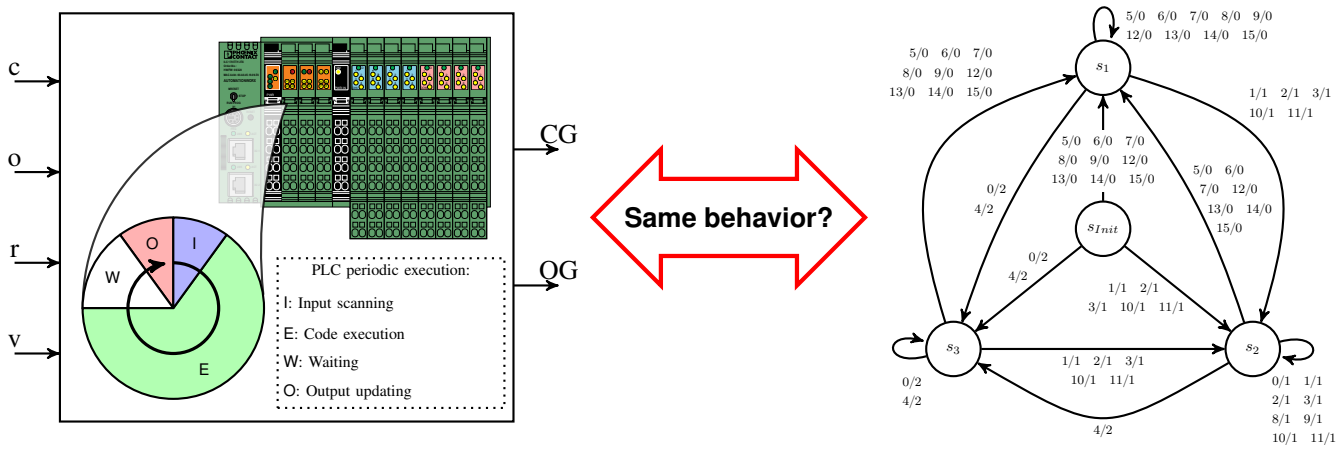

Fig. 2. Aim of the conformance test. Left part: PLC with periodic I/O scanning cycle; right part: Mealy machine describing the specification of the control.

from a Grafcet [30] an equivalent Mealy machine is presented.

Once the formal model of the specification obtained, a *test objective* is to be defined. It is possible, for instance, to test whether some particular states that correspond to hazardous or recovery states can be reached from the initial state or whether some state changes or transition sequences are possible for specific input combinations. When critical systems are considered, as this is the case in this work, a usual test objective is to cross at least once each edge of the directed graph that represents the structure of the machine; this permits to check every state change from each state of the formal model. A test sequence that meets this test objective is termed *complete*.

Then, the test sequence that will be applied to the PLC during the execution of the test can be constructed from the specification model. A test sequence is an ordered list of couples (input combination, expected output combination), termed *test steps*, where the input and expected output combinations correspond respectively to the left and right elements of the label for the considered transition of the Mealy machine; in other words, a test sequence represents the external view of a PLC that executes a control code in conformance with its specification. To avoid tedious, time-consuming and error-prone tasks, the construction of a complete test sequence must be automated; an usual solution is to select the algorithm presented in [31] that minimizes the length of the sequence. A minimum-length test sequence will indeed minimize the duration of the execution of the test, if the duration of each test step is constant.

However, extensive experimental studies have shown in that the execution of a conformance test with a minimum-length test sequence may lead to erroneous test verdicts because synchronous input changes may be detected as asynchronous by the PLC under test [32], [33]. The aim of this paper is to propose another algorithm to construct test sequences. Rather than looking for a minimum-length solution, the overall idea is to construct a test sequence that does not contain synchronous changes of two or more inputs from one test step to the immediately following one. Such a sequence is termed a SIC (Single Input Change) sequence because the value of only one input is modified between two consecutive test steps. It must be noted that the expression *SIC* (or *adjacent*) *sequence* has been already introduced in another domain: test of electronic circuits [34]–[40]. However, the valuable results of these works cannot be directly applied to the issue that we address because they were not focusing on the same type of fault: hardware faults were considered while this work focuses on errors in the PLC code. Moreover, a white-box test was possible in those references whereas the structure of the implementation is unknown in this work (black-box approach); the construction of the test sequence cannot be based on the knowledge of this structure but only of the specification.

Nevertheless, it has not been proved that it is always possible to construct a *complete SIC* sequence *starting from the initial state* of the specification model; this will be the first issue addressed in this paper. Once this issue solved, a solution to construct the SIC sequence will be proposed.

The next section reminds the notations used in this work. The concept of SIC-testability, feature of a Mealy machine that represents the possibility to build an initializable, consistent and complete SIC test sequence from this machine is introduced in the third section; a method to verify whether a Mealy machine is fully SIC-testable as well as to determine the SIC-testable part of a non-fully testable machine is also proposed in this section. The fourth section focuses on the generation of this sequence and provides three solutions according to the connectivity properties of the SIC-testable part. Last, conclusions and prospects for further works are given.

II. Background

The aim of the section is to define the notations used in this paper and to remind previous results. The notations and definitions will be illustrated through an example with 4 logic inputs (c, o, r and v), and 2 logic outputs (OG and CG); then, 16 input combinations (c_I) and 4 output combinations (c_O) can be defined. The PLC where this control is implemented is presented in Fig. 2, left part; the control specification in the form of a Mealy machine is presented in Fig. 2, right part.

A. Notations of the input and output combinations

A PLC owns n logic input variables and m logic output variables; the value of each of them is either *true* or *false*. 2^n input (2^m output) combinations can then be constructed from this set of input (output) variables by assigning a weight to each variable. An input combination c_I will be represented in three different manners in this paper:

- The first representation, noted symbol(c_I), is the more compact one; symbol(c_I) is indeed an integer that belongs to $[0, 2^n - 1]$ and is defined as follows:
  symbol(c_I) = $\sum_{i=0}^{n-1} c_I[n-1-i] \times 2^{(n-1-i)}$, where:
  - $c_I[n-1-i]$ is an integer that belongs to $[0, 1]$ and is equal to 1 if the ith input variable is *true* and 0 otherwise,
  - $2^{(n-1-i)}$ is the weight of this variable[1].

  This representation will be used in the graphical and tabular descriptions of a Mealy machine.
- The second representation, noted minterm(c_I), is a Boolean expression. A minterm is the conjunction of all the n logic input variables in their positive or complemented form. This representation is very efficient for symbolic calculus and will be used to check the SIC-testability of a Mealy machine.
- The third representation, noted $\mathbb{1}(c_I)$, is that of the set of the only variables which are *true* for the given combination and is well appropriate when defining the SIC relation between two combinations.

The same rules apply for the output combinations c_O. Table I gives the correspondence between these representations for the example introduced in Fig. 2.

B. Formal definition of a Mealy machine

Conformance test of Mealy machines is a mature technique that previously yielded numerous sound theoretical results; good syntheses on this topic are available in [25], [41]. This explains why this formalism was selected to represent formally the specification model.

However, a Mealy machine is theoretically defined on two event alphabets: the input and output alphabets. Since the inputs and outputs of a PLC are logic variables and not events, these two alphabets are to be built prior to defining the Mealy machine that represents the specification of the controller. This will be performed by considering each PLC input (respectively output) combination as an input (resp. output) event.

Let us note I and O the non-empty sets of PLC inputs and outputs (I and O contain logic variables). If the cardinality of I (resp. O) is $|I|$ (resp. $|O|$), there exist $2^{|I|}$ (resp. $2^{|O|}$) distinct input (resp. output) combinations c_I (resp. c_O). Let us note C_I the set of the input combinations and C_O the set of the output combinations.

Using this definition of input and output combinations, the specification of a PLC that executes a control code can be represented by a Mealy machine $(S, s_{Init}, C_I, C_O, \delta, \lambda)$, where:

- S is a non-empty set of states.
- s_{Init} is the initial state, $s_{Init} \in S$.
- C_I is the input alphabet, $|C_I| = 2^{|I|}$.
- C_O is the output alphabet, $|C_O| = 2^{|O|}$.
- δ is the transition function, defined as follows:

$$\begin{array}{lll} \delta: & S \times C_I & \rightarrow S \\ & (s_s, c_I) & \mapsto s_t = \delta(s_s, c_I) \end{array} \quad (1)$$

- λ is the output function, defined as follows:

$$\begin{array}{lll} \lambda: & S \times C_I & \rightarrow C_O \\ & (s_s, c_I) & \mapsto c_o = \lambda(s_s, c_I) \end{array} \quad (2)$$

The specification of the controller is compulsorily deterministic: there is only one initial state and δ and λ are two functions. Moreover, to avoid misinterpretation errors during the test, the Mealy machine is:

- completely defined: δ and λ are total functions[2];

$$\forall (s, c_I) \in S \times C_I, \begin{cases} \exists! \delta(s, c_I) \in S \\ \exists! \lambda(s, c_I) \in C_O \end{cases} \quad (3)$$

- limited to its reachable part;

$$\forall s \in S, \ \exists [c_I^0, \cdots, c_I^n] \in C_I^* \mid \begin{cases} s^1 = \delta(s_{Init}, c_I^0) \\ \forall k \geqslant 1, \ s^{k+1} = \delta(s^k, c_I^k) \\ s = s^n \end{cases} \quad (4)$$

- without transient evolution, i.e. no inputs change introduces successive changes of states or emitted outputs.

$$\forall (s, c_I) \in S \times C_I, \begin{cases} \delta(\delta(s, c_I), c_I) = \delta(s, c_I) \\ \lambda(\delta(s, c_I), c_I) = \lambda(s, c_I) \end{cases} \quad (5)$$

C. Formal definition of a test sequence

A test sequence is an ordered list of couples (input combination, expected output combination) which represents the external view of the expected behavior of a PLC that executes a correct control code. Formally, a test sequence is defined as follows:

$$\left[(c_I^0, c_O^0), (c_I^1, c_O^1), ..., (c_I^n, c_O^n)\right] \in (C_I \times C_O)^* \quad (6)$$

However, the input combinations c_I^k and the expected output combinations c_O^k are not independent. The expected output combination c_O is that associated to the transition which goes from a source state s_s to a target state s_t for the input combination c_I. Hence, an elementary conformance test step et is defined by the following 4-tuple:

$$\begin{array}{l} et = (s_s, c_I, s_t, c_O) \in S \times C_I \times S \times C_O \\ \text{where} \begin{cases} s_t = \delta(s_s, c_I) \\ c_O = \lambda(s_s, c_I) \end{cases} \end{array} \quad (7)$$

A test sequence TS is an ordered list of elementary test steps et which must be:

P1: initializable, i.e. the source state of the first test step is the initial state of the PLC's behavior model, and the input combination c_I^0 is such that the target state is stable:

$$\begin{cases} s^0 = s_{Init} \\ \delta(\delta(s^0, c_I^0), c_I^0) = \delta(s^0, c_I^0) \\ \lambda(\delta(s^0, c_I^0), c_I^0) = \lambda(s^0, c_I^0) \end{cases} \quad (8)$$

[1]The weights are assigned arbitrarily to the variables.

[2]$\exists!$: There exists exactly one.

TABLE I
EQUIVALENCE BETWEEN THE DIFFERENT REPRESENTATIONS OF THE INPUT AND OUTPUT COMBINATIONS

(logic inputs, weight)	(c,8)	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	(o,4)	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	(r,2)	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	(v,1)	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
symbol(c_I)		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
minterm(c_I)		$\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}$	$\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v$	$\overline{c}\cdot\overline{o}\cdot r\cdot\overline{v}$	$\overline{c}\cdot\overline{o}\cdot r\cdot v$	$\overline{c}\cdot o\cdot\overline{r}\cdot\overline{v}$	$\overline{c}\cdot o\cdot\overline{r}\cdot v$	$\overline{c}\cdot o\cdot r\cdot\overline{v}$	$\overline{c}\cdot o\cdot r\cdot v$	$c\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}$	$c\cdot\overline{o}\cdot\overline{r}\cdot v$	$c\cdot\overline{o}\cdot r\cdot\overline{v}$	$c\cdot\overline{o}\cdot r\cdot v$	$c\cdot o\cdot\overline{r}\cdot\overline{v}$	$c\cdot o\cdot\overline{r}\cdot v$	$c\cdot o\cdot r\cdot\overline{v}$	$c\cdot o\cdot r\cdot v$
$\mathbb{1}(c_I)$		$\{\}$	$\{v\}$	$\{r\}$	$\{r,v\}$	$\{o\}$	$\{o,v\}$	$\{o,r\}$	$\{o,r,v\}$	$\{c\}$	$\{c,v\}$	$\{c,r\}$	$\{c,r,v\}$	$\{c,o\}$	$\{c,o,v\}$	$\{c,o,r\}$	$\{c,o,r,v\}$

(logic outputs, weight)	(CG,2)	0	0	1	1
	(OG,1)	0	1	0	1
symbol(c_O)		0	1	2	3
minterm(c_O)		$\overline{CG}\cdot\overline{OG}$	$\overline{CG}\cdot OG$	$CG\cdot\overline{OG}$	$CG\cdot OG$
$\mathbb{1}(c_O)$		$\{\}$	$\{OG\}$	$\{CG\}$	$\{CG,OG\}$

P2: consistent, i.e. the source state of the $(k+1)^{th}$ elementary test step is equal to the target state of the kth elementary test step.

$$TS = [(s^0, c_I^0, \delta(s^0, c_I^0), \lambda(s^0, c_I^0)), \cdots, (s^n, c_I^n, \delta(s^n, c_I^n), \lambda(s^n, c_I^n))] \mid \forall k \geqslant 0, s^{k+1} = \delta(s^k, c_I^k) \quad (9)$$

Moreover, if the test objective is to cross at least once each transition of the Mealy machine (usual objective when the control of critical systems is considered), the test sequence must be:

P3: complete, i.e. there is at least one test step for each element of the transition function:

$$\forall (s, c_I) \in (S \backslash s_{Init} \times I),\ (s, c_I, \delta(s, c_I), \lambda(s, c_I)) \in TS \quad (10)$$

III. SIC-TESTABILITY

SIC-testability is a feature of a Mealy machine that represents the possibility to build an initializable, consistent and complete SIC test sequence from this machine. This concept is illustrated in Fig. 3. The example *3a)* is non-SIC-testable because the test step that corresponds to the self-loop on the state s_2 with the input combination $\overline{a}\cdot\overline{b}$ cannot be preceded by a test step with an input combination where only one of the variables a and b is true; both possible preceding test steps correspond to the input combination $a\cdot b$. This non-SIC-testable transition may lead to a biased verdict: if the input change from $a\cdot b$ to $\overline{a}\cdot\overline{b}$ when the machine is in the state s_2 is erroneously interpreted by the PLC, the target state could be either s_2 (as if it was correctly interpreted) or s_1, thus, potentially rejecting a correct implementation. A similar reasoning is possible for the pinpointed transition of *3b)*; the test step that corresponds to the transition from the state s_2 to the state s_1 with the input combination $\overline{a}\cdot\overline{b}$ cannot be preceded by a test step with an input combination where only one of the variables a and b is true; the only possible preceding test step corresponds to the input combination $a\cdot b$. This non-SIC-testable transition may lead to a non-valid verdict: whatever the interpretation (correct or erroneous) of the input change from $a\cdot b$ to $\overline{a}\cdot\overline{b}$, the target state will be s_1. Thus, it cannot be ensured that this specific transition of the implementation has been tested, and an incorrect implementation may be accepted. On the opposite, the example *3c)* is SIC-testable; it is possible to find for any transition a preceding transition whose input combination differs from only one input.

This section proposes first a formal definition of a SIC test sequence, then presents a method to determine the SIC-testable part of a Mealy machine, part of this machine from which such a sequence can be built. If this part contains all test steps that can be defined from the machine, the machine is said fully SIC-testable, else a coverage rate of the test steps can be defined.

A. Definition of a SIC test sequence

A SIC test sequence is an initializable and consistent (relations (8) and (9) satisfied) test sequence that is based on a SIC input sequence. To express formally this latter property, the SIC relation between two input combinations must be first defined. This definition relies on the representation of an input combination by the subset of I that contains the only variables which are *true* for this combination. Thus, two input combinations c_I and c_I' satisfy a SIC relation if and only if[3]:

$$card((\mathbb{1}(c_I)\backslash\mathbb{1}(c_I')) \cup (\mathbb{1}(c_I')\backslash\mathbb{1}(c_I))) = 1 \quad (11)$$

For example, the input combinations which are represented by the minterms $\overline{c}\cdot\overline{o}\cdot r\cdot v$ and $c\cdot\overline{o}\cdot r\cdot v$ satisfy a SIC relation since $card((\{r,v\}\backslash\{c,r,v\}) \cup (\{c,r,v\}\backslash\{r,v\})) = card(\varnothing \cup \{c\}) = 1$

In the remainder of this paper, this symmetrical relation is noted[4]: $c_I\ R_{Gray}\ c_I'$. It must be noted that n SIC relations can be stated for each input combination c_I of a PLC with n logic inputs.

Hence, a test sequence TS is a SIC test sequence if and only if it satisfies the following property:

P5: it is based on a SIC input sequence, i.e.:

$$\forall k > 0,\ c_I^{k+1}\ R_{Gray}\ c_I^k \quad (12)$$

[3] $card(A)$ is the cardinality of set A.
$\mathbb{1}(c_I)\backslash\mathbb{1}(c_I')$ is the subset of I composed with the elements of $\mathbb{1}(c_I)$ which are not in $\mathbb{1}(c_I')$.

[4] The subscript Gray has been introduced because two combinations that satisfy this relation may be seen as two adjacent combinations of a Gray code.

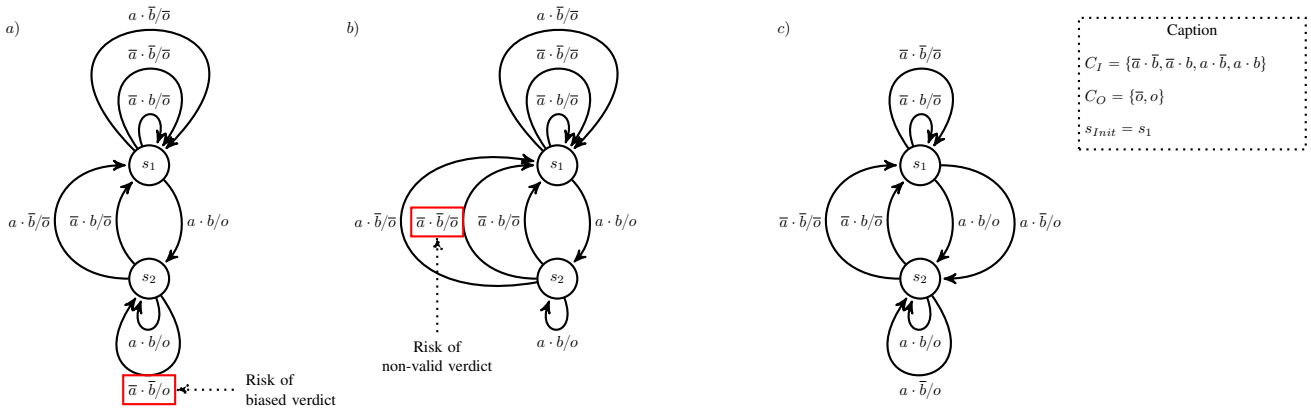

Fig. 3. Examples of non-SIC-testable Mealy machines: a) and b), and a SIC-testable Mealy machine: c)

B. Computation of the SIC-testable part of a Mealy machine

The SIC-testable part of a Mealy machine may be obtained by computing a set R_{SIC} of couples (s_s, c_I), where s_s is the source state of a transition of the Mealy machine and c_I an input combination. Each element of R_{SIC} defines also an elementary test step (s_s, c_I, s_t, c_O) because the target state s_t and the output combination c_O are then completely known from the structure of the machine. The set R_{SIC} is computed iteratively by a fixed point calculation; the set at the ith iteration of this calculation will be noted $R_{SIC}(i)$.

As the SIC sequence must be initializable, the initial set $R_{SIC}(0)$ contains the couples (s_{Init}, c_I) where s_{Init} is the initial state, and c_I is an input combination such that, if s_t is the target state of the transition $(s_{Init}, c_I, s_t, c_O)$, $\delta(s_t, c_I) = s_t$, i.e. there exists a self-loop on s_t for this input combination.

$$R_{SIC}(0) = \left\{(s_{Init}, c_I^0) \mid c_I^0 \in C_I, \delta(\delta(s_{Init}, c_I^0)) = \delta(s_{Init}, c_I^0)\right\} \quad (13)$$

In practice, every logic input of a PLC which is connected to a test bench can be set or reset before the initialization of the PLC. Hence, the state s_t is a state that can become and stay active when the PLC is initialized after a given input combination has been defined.

The following sets $R_{SIC}(k+1)$ are determined by using the two following construction rules:

- If an elementary step (s_t, c_I, s_t, c_O) belongs to a SIC test sequence, it is always possible to add to this sequence an elementary step $(s_t, c_I', \delta(s_t, c_I'), \lambda(s_t, c_I'))$ where c_I' satisfies: $c_I' \; R_{Gray} \; c_I$.
- If an elementary test step (s_s, c_I, s_t, c_O) belongs to a SIC test sequence, the elementary test step (s_t, c_I, s_t, c_O) can be added to this sequence.

These rules can be formally expressed by the following statement:

$$R_{SIC}(k+1) = R_{SIC}(k) \cup \left\{ \left(s_k, c_I^{k+1}\right) \cup \left(\delta\left(s_k, c_I^{k+1}\right), c_I^{k+1}\right) \mid \exists (s_k, c_I^k) \in R_{SIC}(k) \mid \left\{ \begin{array}{l} \delta(s_k, c_I^k) = s_k \\ c_I^{k+1} \; R_{Gray} \; c_I^k \end{array} \right\} \right. \quad (14)$$

The computation stops when $R_{SIC}(k+1) = R_{SIC}(k)$. The Mealy machine is then SIC-testable if $R_{SIC}(k+1)$ contains as many couples as there are potential test steps. Otherwise, the final set $R_{SIC}(k+1)$, denoted R_{SIC}^{Maxi}, defines the SIC-testable part. A SIC coverage rate, defined as follows, permits to quantify the SIC-testability:

$$\text{SIC coverage rate} = \frac{|R_{SIC}^{Maxi}|}{|S \backslash s_{Init}| \times |C_I|} \quad (15)$$

This rate can be seen as a metrics of the ability of the specification to be used to build a complete SIC test sequence. Improving the coverage rate requires the specification be modified, which is not always possible for cost and time reasons.

C. Illustration on the example

Table II presents the results of this calculation for the example presented in Fig. 2. Each cell of the table contains the value of the couple $(\delta(s, c_I), \lambda(s, c_I))$. A circled couple means that the same state is both source and target of the transition (self-loop structure: $\delta(s, c_I) = s$). The behavior represented in this table is deterministic and completely defined since every cell contains one and only one state name. This behavior does not contain any transient evolution since the value of each cell is either a circled value or leads to a cell with a circled value ($\delta(s, c_I) = s$ or $\delta(\delta(s, c_I), c_I) = \delta(s, c_I)$). The number k of the iteration during which the couple (s_s, c_I) was added to $R_{SIC}(k)$ is at the top-left corner of each cell. For example, the couple associated to the cell $(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$ is obtained at iteration 0 (initialization), because s_3 is reachable from s_{Init} $(\delta(s_{Init}, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}) = s_3)$, and this couple corresponds to a self-loop on s_3 $(\delta(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}) = s_3)$. The couple $(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v)$ is obtained in the first iteration, as $\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v \; R_{Gray} \; \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}$, and so on. The fixed-point calculation stops at the third iteration, excluding the initialization. The final set contains only 40 couples; the couples that do not belong to this set are represented by colored cells. Hence, the Mealy machine of Fig. 2 is not fully SIC-testable. Its SIC-testable part R_{SIC}^{Maxi} is represented by the cells that are not colored. Its SIC coverage rate is equal to 5/6.

TABLE II
ILLUSTRATION OF THE FIXED POINT CALCULATION. IN THIS TABLE, THE OUTPUT COMBINATIONS ARE OMITTED FOR CLARITY REASONS.

source state s_s \ minterm(c_I)	$\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}$	$\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v$	$\overline{c}\cdot\overline{o}\cdot r\cdot\overline{v}$	$\overline{c}\cdot\overline{o}\cdot r\cdot v$	$\overline{c}\cdot o\cdot\overline{r}\cdot\overline{v}$	$\overline{c}\cdot o\cdot\overline{r}\cdot v$	$\overline{c}\cdot o\cdot r\cdot\overline{v}$	$\overline{c}\cdot o\cdot r\cdot v$	$c\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}$	$c\cdot\overline{o}\cdot\overline{r}\cdot v$	$c\cdot\overline{o}\cdot r\cdot\overline{v}$	$c\cdot\overline{o}\cdot r\cdot v$	$c\cdot o\cdot\overline{r}\cdot\overline{v}$	$c\cdot o\cdot\overline{r}\cdot v$	$c\cdot o\cdot r\cdot\overline{v}$	$c\cdot o\cdot r\cdot v$
s_{Init}	s_3	s_2	s_2	s_2	s_3	s_1	s_1	s_1	s_1	s_1	s_2	s_2	s_1	s_1	s_1	s_1
s_1	$^1s_3$	$^1s_2$	$^1s_2$	$^1s_2$	$^1s_3$	$^0s_1$	$^0s_1$	$^0s_1$	$^0s_1$	$^0s_1$	$^1s_2$	$^1s_2$	$^0s_1$	$^0s_1$	$^0s_1$	$^0s_1$
s_2	$^1s_2$	$^0s_2$	$^0s_2$	$^0s_2$	$^2s_3$	$^1s_1$	$^1s_1$	$^1s_1$	$^1s_2$	$^1s_2$	$^0s_2$	$^0s_2$	$^2s_1$	$^2s_1$	$^1s_1$	$^1s_1$
s_3	$^0s_3$	$^1s_2$	$^1s_2$	s_2	$^0s_3$	$^1s_1$	$^1s_1$	s_1	$^1s_1$	s_1	s_2	s_2	$^1s_1$	s_1	s_1	s_1

D. Symbolic computation of the SIC-testable part

As already mentioned, the tabular representation of a Mealy machine is appropriate to explain the principle of computations on small-sized models but is not suitable to perform these computations on non-trivial models. This explains why a symbolic representation of a set of input combinations has been introduced to avoid explicit enumeration of this set during the fixed point calculation.

A set C of combinations c can be represented by a Boolean expression $Exp(C)$ defined as the disjunction of the minterms contained in C:

$$Exp(C) = \mathop{+}_{c\in C} \text{minterm}(c) \tag{16}$$

During the fixed-point calculation, a set of input combinations C can be extended by adding all input combinations c'_I satisfying a SIC relation with at least one of the input combinations c_I in C. The extended set C' is defined as follows:

$$C' = C \cup \{c'_I \mid \exists c_I \in C :\ c'_I\ R_{Gray}\ c_I\}, \tag{17}$$

Using symbolic notation, the Boolean expression of the extended set C' is defined as follows:

$$Exp(C') = \mathop{+}_{i\in I} \left(Exp(C)_{|i\leftarrow false} + Exp(C)_{|i\leftarrow true}\right) \tag{18}$$

The example below illustrates this operation.

$$\begin{aligned}
C &= \{\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v\,,\ \overline{c}\cdot\overline{o}\cdot r\cdot\overline{v}\,,\ \overline{c}\cdot\overline{o}\cdot r\cdot v\}\\
Exp(C) &= \overline{c}\cdot\overline{o}\cdot r + \overline{c}\cdot\overline{o}\cdot v\\
Exp(C') &= (\overline{o}\cdot r + \overline{o}\cdot v) + (\overline{c}\cdot r + \overline{c}\cdot v) + (\overline{c}\cdot\overline{o})\\
&\quad +(\overline{c}\cdot\overline{o})\\
&= \overline{c}\cdot\overline{o} + \overline{c}\cdot r + \overline{c}\cdot v + \overline{o}\cdot r + \overline{o}\cdot v\\
C' &= \{\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}\ ,\ \overline{c}\cdot\overline{o}\cdot r\cdot\overline{v}\ ,\ c\cdot\overline{o}\cdot r\cdot\overline{v}\ ,\\
&\quad \overline{c}\cdot o\cdot r\cdot\overline{v}\ ,\ \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v\ ,\ c\cdot\overline{o}\cdot\overline{r}\cdot v\ ,\\
&\quad \overline{c}\cdot o\cdot\overline{r}\cdot v\ ,\ \overline{c}\cdot\overline{o}\cdot r\cdot v\ ,\ c\cdot\overline{o}\cdot r\cdot v\ ,\\
&\quad \overline{c}\cdot o\cdot r\cdot v\}
\end{aligned} \tag{19}$$

This symbolic representation of a set of input combinations speeds up the computation defined in the equation (14).

IV. SIC TEST SEQUENCE GENERATION

Once R_{SIC}^{Maxi} determined, a SIC test sequence can be constructed by using a graphical representation of this set in the form of a graph where:

- each node represents a couple (s, c_I) that is included in R_{SIC}^{Maxi};
- n arcs start from a node that corresponds to a couple (s, c_I) such that $\delta(s, c_I) = s$. The target nodes of these arcs represent the couples (s, c'_I) such that c'_I satisfies $c'_I\ R_{Gray}\ c_I$. These arcs correspond to input changes between two test steps; the cost associated to these arcs is then equal to 1;
- only one arc starts from a node that corresponds to a couple (s, c_I) such that $\delta(s, c_I) \neq s$. The target node of this arc is the node that represents the couple $(\delta(s, c_I), c_I)$. This arc corresponds to the expected evolution from a source state to a target state during the execution of one test step; the cost associated to this arc is then equal to 0.

Fig. 4 represents a part of this graph. Each node corresponds to a couple (s_s, c_I) where s_s corresponds to its line and c_I corresponds to its row. Since the couples $(s_3, \overline{c}\cdot\overline{o}\cdot r\cdot v)$ and $(s_3, \overline{c}\cdot o\cdot r\cdot v)$ are not in R_{SIC}^{Maxi} there is no node associated to these couples. In this figure, only the arcs related to the couple $(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$ are represented. Since this couple can be tested in the same experimental test step than $(s_{Init}, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$, there is one arc from $(s_{Init}, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$ leading to $(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$. From this couple, there are four outgoing arcs leading to couples $(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v)$, $(s_3, \overline{c}\cdot\overline{o}\cdot r\cdot\overline{v})$, $(s_3, \overline{c}\cdot o\cdot\overline{r}\cdot\overline{v})$ and $(s_3, c\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$. Then, since the couple $\delta(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v) = s_2$, there is one outgoing arc from $\delta(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v)$ to $\delta(s_2, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v)$. On the opposite, since $\delta(s_3, \overline{c}\cdot o\cdot\overline{r}\cdot\overline{v}) = s_3$, there are four outgoing arcs from $(s_3, \overline{c}\cdot o\cdot\overline{r}\cdot\overline{v})$.

A SIC test sequence can be then constructed by looking for a path that traverses at least once each node of this graph. To reduce the duration of test execution, a minimum-length SIC test sequence can be searched; the optimization problem to solve in this case is a particular solution of a well-known problem in graph theory: the Travelling Salesman Problem [42] – or pre-Hamiltonian path –. The general formulation of this problem is the following: Find a minimum-length closed path that traverses at least once each **node** of the graph. By

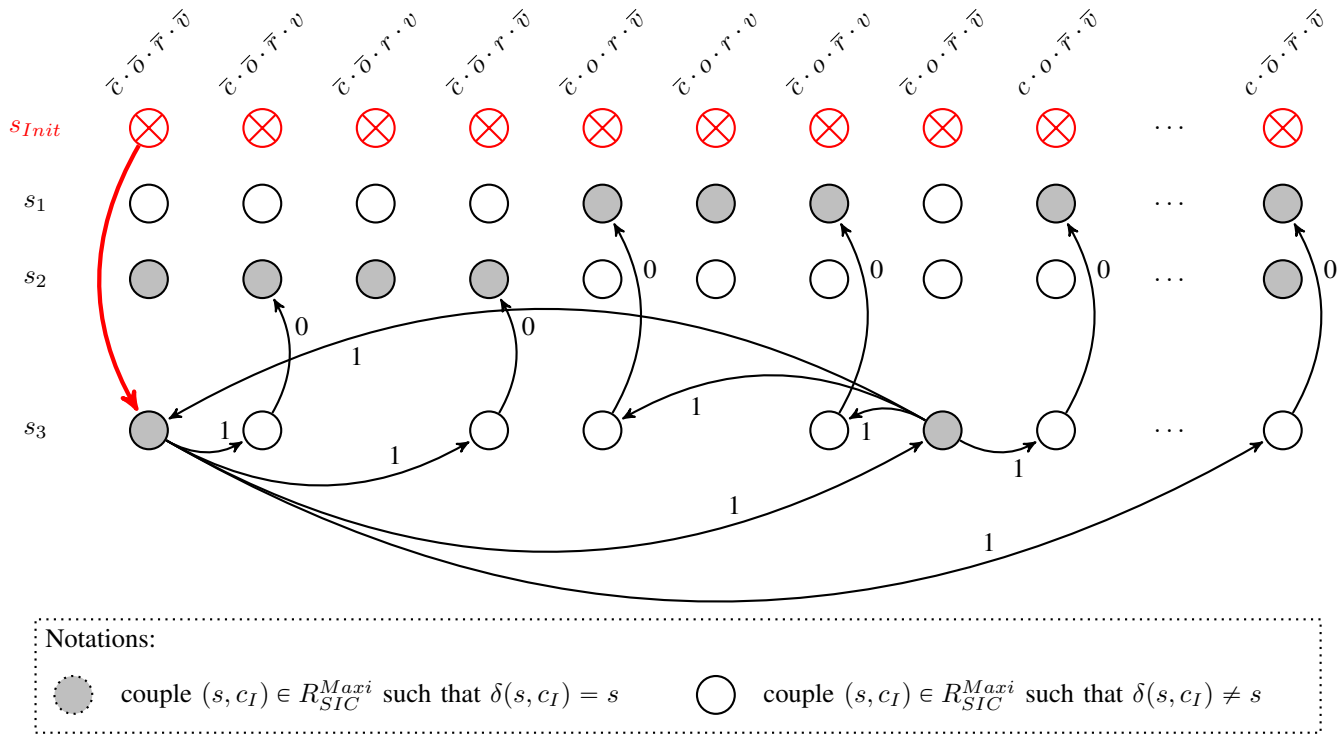

Fig. 4. Part of the graph used to generate a SIC test sequence

TABLE III
SIC TEST SEQUENCE FOR THE SIC-TESTABLE PART OF THE EXAMPLE

s_s:	s_1	s_1	s_3	s_2	s_2	s_3	s_3	s_2	s_1	s_1	s_2	s_1	s_1	s_2	s_2	s_1	s_1	s_2	s_1	s_2	s_2	s_1	s_2	s_2	s_1	s_3	s_1	s_3	s_1	s_2	s_1	s_3	s_1	s_3	s_3
c:	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	1
o:	0	0	0	0	1	0	0	1	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	1	1	1	1	0	1	1	1	1	0	0
r:	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0
v:	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
s_t:	s_1	s_3	s_2	s_2	s_3	s_3	s_2	s_1	s_1	s_2	s_1	s_1	s_2	s_2	s_1	s_1	s_2	s_1	s_2	s_2	s_1	s_2	s_2	s_1	s_3	s_1	s_3	s_1	s_2	s_1	s_3	s_1	s_3	s_3	s_1

definition, the length between two nodes is equal to the sum of the costs associated to the collection of arcs that define the shortest path between these two nodes.

However, possible to construct a single SIC test sequence that traverses each node at least once and starts from the initial state with any input combination if and only if the graph that represents R_{SIC}^{Maxi} is strongly connected[5]. Then, a strategy to construct SIC test sequences whatever the connectivity of the graph has been set up (see Fig. 5). If the graph is only connected (and not strongly connected), a single SIC test sequence can be constructed but this sequence must start by an elementary test step that contains a particular input combination. When the graph is not connected, several SIC test sequences shall be constructed; during test execution, the PLC shall be initialized between two of these sequences because each of them starts from the initial state by definition (relation (8)).

For the example presented in Fig. 2 it is possible to generate a single SIC-test sequence that covers its SIC-testable part. This sequence is given in Table III where the top and bottom lines have been added to relate this sequence to Fig. 2 and Table II. This test sequence contains 35 test steps and permits to test the 40 couples (s, c_I) of the SIC-testable part of the specification since some test steps permit to test both couples (s, c_I) and $(\delta(s, c_I), c_I)$, as already mentioned; for example, test step 2 permits to test both $(s_1, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$ and $(s_2, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$, and so on for all test steps whose source and target states are different. The test sequence given in Table III is obtained in approximately 4 seconds; this computation lasts longer than that of a minimum-length sequence because the problem to solve is harder.

[5]A graph is strongly connected if and only if it contains a path from n_i to n_j and a path from n_j to n_i for every pair of nodes n_i, n_j

V. CONCLUSION

Even if numerous theoretical results on conformance test of Mealy machines have been published, application to conformance test of PLCs is not completely straightforward because the technological features of these industrial components are not taken into account in the theoretical studies, as pinpointed in [32], [33].

A promising solution to tackle out biased or non-valid test results due to asynchronism between input events that are assumed to be synchronous is to construct a test sequence, termed *SIC test sequence*, where no synchronous input events are present by definition. This paper has presented how the part

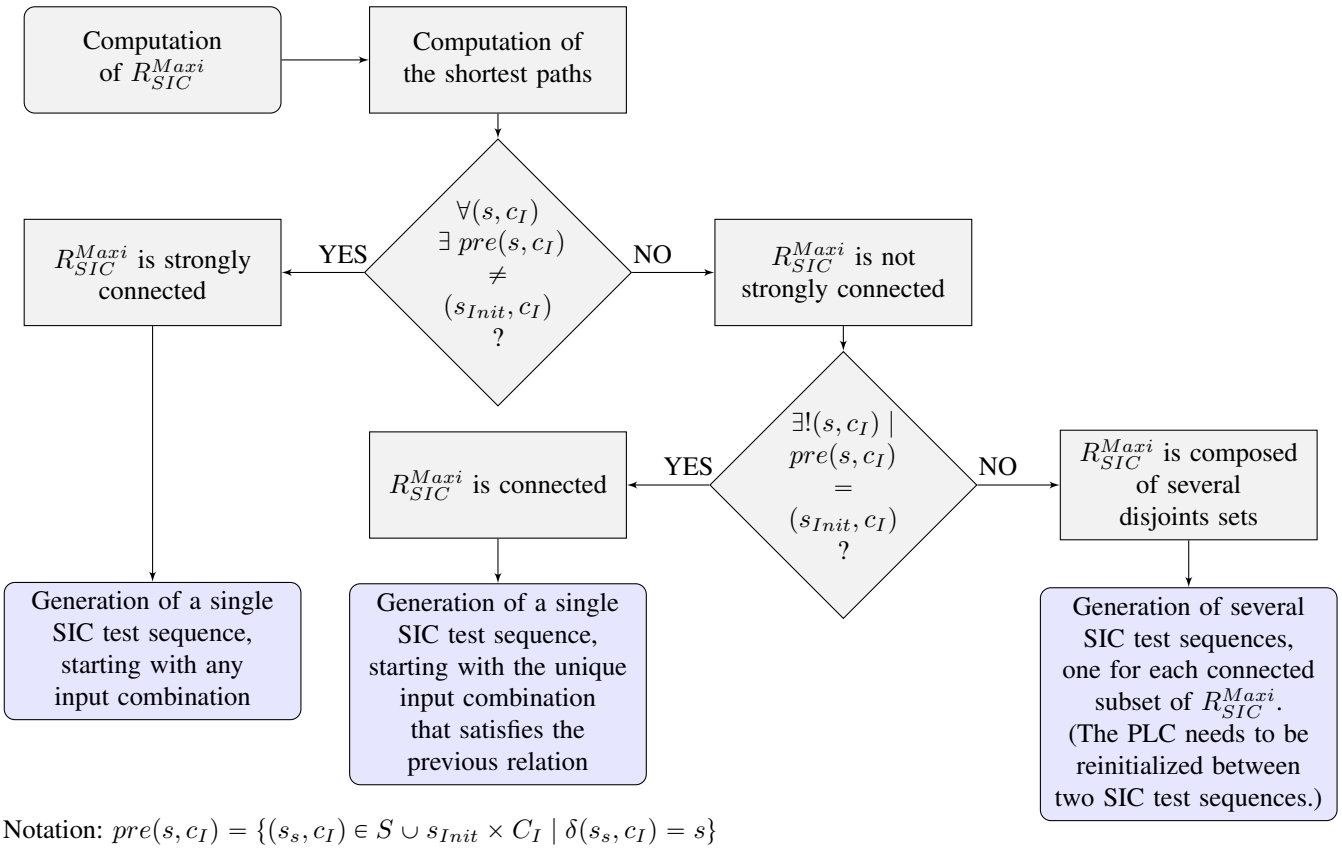

Fig. 5. Flowchart showing the different cases to consider when generating a SIC test sequence

of the Mealy machine that can be tested with such a sequence can be determined and how to construct this sequence.

A coverage rate has been also defined. This rate is however not always equal to 100%. If the objective of the conformance test is to test every transition of the Mealy machine, it will be necessary to use a non-SIC sequence for the transitions which cannot be tested with the SIC sequence. The following stategies shall be then considered:

- test execution for the configurations of the controller that lessen the error rate (periodic I/O scanning and no inputs distribution) as shown in [32], [33];
- multiple execution of the same test sequence and statistical analyses of the results.

Further works are aiming at extending the scope of this study by considering construction of test sequences for timed systems – the formal model that will be used to build this sequence will be a class of timed automata – and analysis approaches based on discrete event systems theory that are complementary to conformance test, like identification or enforcement, to validate the behavior of a PLC.

References

[1] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 3, pp. 1234–1249, 2013.

[2] R. Drath, A. Luder, J. Peschke, and L. Hundt, "AutomationML - the glue for seamless automation engineering," in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*. IEEE, 2008, pp. 616–623.

[3] E. Estévez and M. Marcos, "Model based validation of industrial control systems," *Industrial Informatics, IEEE Transactions on*, vol. 8, no. 2, pp. 302–310, 2012.

[4] M. Witsch and B. Vogel-Heuser, "Towards a formal specification framework for manufacturing execution systems," *Industrial Informatics, IEEE Transactions on*, vol. 8, no. 2, pp. 311–320, 2012.

[5] M. Wehrmeister, C. Pereira, and F. Rammig, "Aspect-oriented model-driven engineering for embedded systems applied to automation systems," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 4, pp. 2373–2386, 2013.

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

[7] S. Patil, V. Vyatkin, and M. Sorouri, "Formal verification of intelligent mechatronic systems with decentralized control logic," in *17th International Conference on Emerging Technologies and Factory Automation (ETFA 2012)*, 2012.

[8] M. Perin and J.-M. Faure, "Building meaningful timed models of closed-loop DES for verification purposes," *Control Engineering Practice*, 2012.

[9] S. Preuße, H.-C. Lapp, and H.-M. Hanisch, "Closed-loop system modeling, validation, and verification," in *17th International Conference on Emerging Technologies and Factory Automation (ETFA 2012)*, 2012.

[10] C. Seidner and O. H. Roux, "Formal methods for systems engineering behavior models," *IEEE Transactions on Industrial Informatics*, vol. 4, no. 4, pp. 280–291, 2008.

[11] V. Vyatkin and H.-M. Hanisch, "A modeling approach for verification of IEC 1499 function blocks using net condition/event systems," in *Emerging Technologies and Factory Automation, 1999. Proceedings. ETFA'99. 1999 7th IEEE International Conference on*, vol. 1. IEEE, 1999, pp. 261–270.

[12] X. Weng and L. Litz, "Verification of logic control design using SIPN and model checking: methods and case study," in *American Control Conference, 2000. Proceedings of the 2000*, vol. 6. IEEE, 2000, pp. 4072–4076.

[13] S. Klein, G. Frey, J.-J. Lesage, and L. Litz, "Supporting the changeability

of SIPN-based logic control algorithms by verification and validation," in *Proc. of IMACS-IEEE int. conf. on Computational Engineering in Systems Applications*, 2003.

[14] G. Canet, S. Couffin, J.-J. Lesage, A. Petit, and P. Schnoebelen, "Towards the automatic verification of PLC programs written in Instruction List," in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, vol. 4. IEEE, 2000, pp. 2449–2454.

[15] S. Lampérière-Couffin and J.-J. Lesage, "Formal verification of the sequential part of PLC programs," in *Workshop on Discrete Event Systems (WODES'00)*. Springer, 2000, pp. 247–254.

[16] V. Gourcuff, O. de Smet, and J.-M. Faure, "Efficient representation for formal verification of PLC programs," in *Proceedings of 8th International Workshop On Discrete Event Systems (WODES'06)*, Ann Arbor USA, 07 2006, pp. pp. 182–187.

[17] G. Čengić and K. Åkesson, "On formal analysis of IEC 61499 applications, part a: Modeling," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 2, pp. 136–144, 2010.

[18] ——, "On formal analysis of IEC 61499 applications, part b: Execution semantics," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 2, pp. 145–154, 2010.

[19] D. Soliman, K. Thramboulidis, and G. Frey, "Transformation of function block diagrams to UPPAAL timed automata for the verification of safety applications," *Annual Reviews in Control*, 2012.

[20] IEC 60880, *Nuclear power plants - Instrumentation and control systems important to safety - Software aspects for computer-based systems performing category A functions*, 2nd ed. International Electrotechnical Commission, 2006.

[21] IEC 61850-10, *Communications Networks and Systems in Substations - Part 10: Conformance testing*, 2nd ed. International Electrotechnical Commission, 2005.

[22] D. Maclay, "Simulation gets into the loop," *IEE Review*, vol. 43, no. 3, pp. 109–112, 1997.

[23] F. Gu, W. S. Harrison, D. M. Tilbury, and C. Yuan, "Hardware-in-the-loop for manufacturing automation control: Current status and identified needs," in *Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on*. IEEE, 2007, pp. 1105–1110.

[24] N. Schetinin, N. Moriz, B. Kumar, A. Maier, S. Faltinski, and O. Niggemann, "Why do verification approaches in automation rarely use HIL-test?" in *Industrial Technology (ICIT), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1428–1433.

[25] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines - a survey," in *Proceedings of the IEEE*, vol. 84, no. 8, 1996, pp. 1090–1123.

[26] J. Tretmans, "Model based testing with labelled transition systems," in *Formal Methods and Testing*, ser. Lecture Notes in Computer Science, R. M. Hierons, J. P. Bowen, and M. Harman, Eds. Springer, 2008, vol. 4949, pp. 1–38.

[27] S. Pickin, C. Jard, T. Jéron, J.-M. Jézéquel, and Y. Le Traon, "Test synthesis from UML models of distributed software," *Software Engineering, IEEE Transactions on*, vol. 33, no. 4, pp. 252–269, 2007.

[28] M. Krichen and S. Tripakis, "Conformance testing for real-time systems," *Formal Methods in System Design*, vol. 34, no. 3, pp. 238–304, 2009.

[29] J. Provost, J.-M. Roussel, and J.-M. Faure, "Translating Grafcet specifications into Mealy machines for conformance test purposes," *Control Engineering Practice*, vol. 19, no. 9, pp. 947–957, 2011.

[30] IEC 60848, *GRAFCET specification language for sequential function charts*, 2nd ed. International Electrotechnical Commission, 2002.

[31] S. Naito and M. Tsunoyama, "Fault detection for sequential machines by transitions tours," in *Proceedings of the IEEE Fault Tolerant Computer Symposium*, 1981, pp. 238–243.

[32] J. Provost, J.-M. Roussel, and J.-M. Faure, "Testing programmable logic controllers from finite state machines specification," in *Proceedings of the 3rd Workshop on Dependable Control of Discrete Systems (DCDS'11)*, 2011.

[33] ——, "Conformance test of programmable logic controllers — execution of minimum-length test sequences," École Normale Supérieure de Cachan, France, Tech. Rep., 2014. [Online]. Available: http://www.lurpa.ens-cachan.fr/-244795.kjsp

[34] M. A. Gharaybeh, M. L. Bushnell, and V. D. Agrawal, "Classification and test generation for path-delay faults using single struck-at fault tests," *Journal of Electronic Testing*, vol. 11, no. 1, pp. 55–67, 1997.

[35] R. David, P. Girard, C. Landrault, S. Pravossoudovitch, and A. Virazel, "Hardware generation of random single input change test sequences," *Journal of Electronic Testing*, vol. 18, no. 2, pp. 145–157, 2002.

[36] A. Virazel, R. David, P. Girard, C. Landrault, and S. Pravossoudovitch, "Delay fault testing: Choosing between random SIC and random MIC test sequences," *Journal of Electronic Testing: Theory and Applications*, vol. 17, no. 3-4, pp. 233–241, 2001.

[37] I. Voyiatzis, T. Haniotakis, and C. Halatsis, "Algorithm for the generation of SIC pairs and its implementation in a BIST environment," *IEE Proceedings-Circuits, Devices and Systems*, vol. 153, no. 5, pp. 427–432, 2006.

[38] W. Yi, F. Xing-hua, and W. Dai-qiang, "An implementation of random single input change technique for low-power test," in *Proceeding of the 2nd International Conference on Anti-counterfeiting, Security and Identification*, 2008, pp. 352–355.

[39] B. Ye, T. Li, Q. Zhao, D. Zhou, X. Wang, and M. Luo, "A low power test pattern generation for built-in self-test based circuits," *International Journal of Electronics*, vol. 98, no. 3, pp. 301–309, 2011.

[40] F. Liang, L. Zhang, S. Lei, G. Zhang, K. Gao, and B. Liang, "Test patterns of multiple SIC vectors: theory and application in BIST schemes," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 4, 2013.

[41] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, Eds., *Model-Based Testing of Reactive Systems, Advanced Lectures*. Springer, 2005, vol. 3472 of Lecture Notes in Computer Science.

[42] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, 1954.

Julien Provost Julien Provost received the Ph.D. degree from École Normale Supérieure de Cachan, France, in 2011. Then, he joined Chalmers University of Technology, Sweden, as a Post-Doctoral Researcher for two years. Currently, he is Assistant Professor at Technische Universität München, Germany where he holds the Assistant Professorship for Safe Embedded Systems. His research interest focuses on the development of formal methods for specification, verification and validation of safe-critical distributed Discrete Event Systems (DES).

Jean-Marc Roussel Jean-Marc Roussel received the Ph.D. degree in 1994. He is currently Associate Professor of Automatic Control at École Normale Supérieure de Cachan (France). His research fields are modeling, synthesis and analysis of control systems with formal methods.

Jean-Marc Faure Jean-Marc Faure received the Ph.D. degree from École Centrale de Paris in 1991. He is currently Professor of Automatic Control and Automation Engineering at the Institut Superieur de Mecanique de Paris and researcher at École Normale Superieure de Cachan, France. His research fields are modeling, synthesis and analysis of Discrete Event Systems (DES) with special focus on formal verification and conformance test methods to improve dependability of critical systems.

J.-M. Faure is member of the IEEE and Associate Editor of the Journal T-ASE since 2012. He is chair of the steering committee of the IFAC workshop series "Dependable Control of Discrete Systems" and has served in many committees of IFAC and IEEE conferences.