SIC-testability of sequential logic controllers

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

SIC-testability of sequential logic controllers ⋆

J. PROVOST J.-M. ROUSSEL J.-M. FAURE

LURPA, ENS Cachan,
61 Avenue du Président Wilson, 94235 Cachan Cedex, France.
(e-mail: {provost, roussel, faure}@lurpa.ens-cachan.fr)

Abstract:
SIC (Single Input Change) test sequences must be privileged for conformance test of logic controllers, to prevent from erroneous test results when the test-bench and the implementation under test are not synchronized. This paper proposes first a definition of the SIC-testable part of a sequential specification model, i.e. the part of the model that can be tested by using a sequence starting from the initial state and for which only one input can change at one at the same time. Then, an algorithm to determine the SIC-testable part is given; if this part is the whole specification, the specification model is declared totally SIC-testable. Once the SIC-testable part obtained, a SIC sequence for conformance test of an implementation of this part can be generated. These contributions are exemplified on an example.

Keywords: Conformance test, Model-based test, Test sequence generation, Logic systems, Testability criterion

1. INTRODUCTION

Logic controllers are increasingly used in critical systems, like power production and distribution systems or transport systems, even for safety-related functions. To ensure dependability of these systems, it really matters to check, before operation, whether each controller behaves correctly with respect to its specification. This is the aim of conformance test, a black-box test technique that is experimentally performed (Figure 1) by sending to the controller an input sequence and comparing the observed output sequence, controller's response to the input sequence, to the expected output sequence so as to build the test verdict: the implemented controller is conform or not. The set of the input sequence and expected output sequence is termed test sequence.

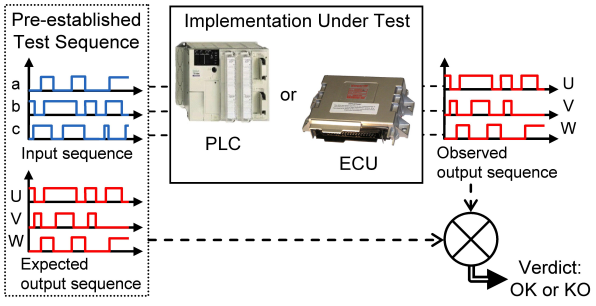


Fig. 1. Conformance test execution

In industrial practice, test sequences are often built manually on the basis of the designer's experience and skill; this is a very tedious, time-consuming and error-prone task.

⋆ This work is funded by the French National Research Agency (TESTEC project, Ref. TLOG 07-022)

To solve this issue, it is possible to take benefit from the numerous theoretical results that have been published in the domain of conformance test, assuming that the specification is formally described, for instance in the form of a finite state machine (Lee and Yannakakis (1996), da Silva Simão et al. (2009)), a transition system (Tretmans (2008)) or a particular class of Petri net (von Bochmann and Jourdan (2009)). (Provost et al. (2009)), for instance, presents a method to translate a specification described in an industrial standardized language into a finite state machine to take benefit from the results on conformance test of these models. Generally speaking, these works provide a way to build automatically a well-defined test sequence from the specification model and to deliver a verdict from the observed output sequence. The test sequence is constructed so as to meet a test objective that is formally defined: exhaustiveness (all transitions are visited at least once) and minimum-length, non-occurrence of unexpected sequence of events, reachability of states, ... The SIC (Single Input Change) or MIC (Multiple Input Change) feature of the input sequence is never considered, though.

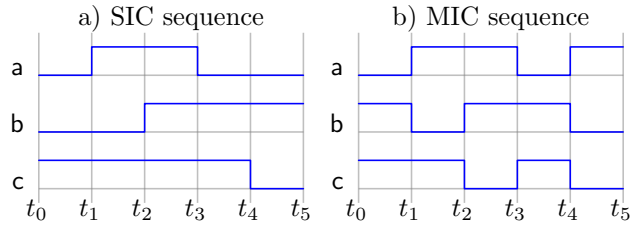


Fig. 2. Example of SIC and MIC test sequences

In a SIC input sequence, named SIC sequence in the remainder of this paper for simplicity reasons, (Figure 2),

the value of only one input can change at one and the same time; synchronous values changes are not allowed. This is not the case in a MIC input sequence. These concepts are used for electronics circuits testing where SIC sequences are privileged to limit power consumption (Yi et al. (2008)) or to increase the efficiency of delay fault testing (Virazel et al. (2001)). Power consumption limitation is not a significant concern during the test of a logic controller and only non-timed systems are addressed in this work. SIC sequence generation is to be investigated however because such a sequence prevents from erroneous test results (biased results if a conform implementation is declared non-conform or non-valid results if implementation errors are not detected) when the test-bench and the logic controller are not synchronized. In this case indeed, when a MIC input sequence is generated by the test-bench, synchronous inputs changes may be seen asynchronous by the controller under test. For technological reasons, all inputs are not read simultaneously by the controller and, in case of two synchronous changes, it may get the value of one input just before its change and the value of another input just after its change. In figure 2 b), the simultaneous changes of a and b at date t_1 that are issued from the test-bench may be interpreted by the controller in one of the following fashions:

- Simultaneous changes of a and b;
- Change of a just followed by change of b;
- Change of b just followed by change of a.

It could be argued that a simple solution is to synchronize the bench and the controller; this is not technically so simple and often impossible if non-invasive test is compulsory. This explains why generation of SIC sequences deserves to be investigated.

It is always possible and easy to build a SIC sequence for a combinatorial logic system by arranging the inputs values according to the Gray code. This is no more the case when sequential systems are focused on; in this case indeed, the aim of the test sequence is to explore a part or the totality of the evolutions and some evolutions may require simultaneous inputs changes. Then, the aim of this work is to propose a method to determine the SIC-testable part of the specification model, i.e. the part that can be explored by using a SIC sequence which starts from the initial state. If this part corresponds to the whole specification, the specification is declared totally SIC-testable. Once the SIC-testable part defined, a SIC sequence for conformance test of an implementation of this part can be built.

Analysis of the SIC-testability requires a formal definition of a test sequence that relies on inputs-outputs values. This implies to model the behavior of the logic controller on the same bases. This behavioral model is proposed in section 2, while the model and properties of the test sequence are given in section 3. Section 4 shows how the SIC-testable part can be obtained by a fixed-point calculation and section 5 deals with SIC sequence construction. Some prospects for further research are sketched in section 6.

2. MODELING THE BEHAVIOR OF SEQUENTIAL LOGIC CONTROLLERS

Several event-based formalisms have been proposed to model sequential systems; they have permitted to obtain worthwhile theoretical results in the domains of synthesis and analysis of DES. However, when design, implementation and test of sequential logic controllers are considered, it is more convenient to select a modeling formalism whose inputs and outputs are logic variables, and not events, and that allows a behavior be represented by a state machine whose transition conditions depend on combinations of inputs values. This section aims to define this formal model.

2.1 Definitions

Formally, a Sequential Logic Controller (SLC) is defined by a 3-tuple $(I, O, \mathcal{B})$ where:

- I is a non-empty set of logic inputs.
- O is a non-empty set of logic outputs.
- $\mathcal{B}$ is the behavior of the controller.

As inputs are logic variables and not events, they can be combined together to give inputs valuations. If the cardinality of I is $|I|$, there exist $2^{|I|}$ distinct inputs valuations v_I. Let us note V_I the set of the inputs valuations and V_O the set of the outputs valuations. As an inputs valuation of a given set I of logic inputs is a Boolean combination, it can be represented by a minterm [1] .

A sequence of inputs valuations is an ordered list of inputs valuations and will be noted as follows:

$$\left[v_I^0, v_I^1, \cdots, v_I^n\right] \in V_I^* \tag{1}$$

The behavior $\mathcal{B}$ of a SLC is defined by a 4-tuple $(S, s_{Init}, \delta, \lambda)$, where:

- S is a non-empty set of states.
- s_{Init} is the initial state, $s_{Init} \in S$.
- δ is the transition function, defined as follows:

$$\begin{aligned} \delta : S \times V_I &\rightarrow S \\ (s, v_I) &\mapsto s' = \delta(s, v_I) \end{aligned} \tag{2}$$

- λ is the output function, defined as follows:

$$\begin{aligned} \lambda : S &\rightarrow V_O \\ s &\mapsto v_o = \lambda(s) \end{aligned} \tag{3}$$

2.2 Properties of a SLC

By definition, the behavior $\mathcal{B}$ of a SLC is deterministic: there is only one initial state and δ and λ are two functions. Moreover, to avoid misinterpretation errors during the test, the behavior $\mathcal{B}$ must be:

- completely defined: δ and λ are total functions [2] (equation 4);
- reduced to its only reachable part (equation 5);
- without transient evolution (equation 6), i.e. no inputs change introduces successive changes of states.

[1] A minterm is the product of all the n Boolean input variables in the positive or complemented form.
[2] $\exists!$: There exists exactly one.

$$\forall (s, v_I) \in S \times V_I, \ \begin{cases} \exists! \delta(s, v_I) \in S \\ \exists! \lambda(s) \in V_O \end{cases} \quad (4)$$

$$\forall s \in S,$$
$$\exists [v_I^0, \cdots, v_I^n] \in V_I^* \mid \begin{cases} s^1 = \delta(s_{Init}, v_I^0) \\ \forall k > 1, \ s^k = \delta(s^{k-1}, v_I^{k-1}) \\ s = s^n \end{cases} \quad (5)$$

$$\forall (s, v_I) \in S \times V_I, \ \delta(s, v_I) = \delta(\delta(s, v_I), v_I) \quad (6)$$

2.3 Illustration on an example

The above definitions and properties will be illustrated on the example of figure 3. This controller owns 4 inputs (c, o, r and v), and 2 outputs (CG and OG); then, 16 inputs valuations (v_I) and 4 outputs valuations (v_O) can be defined. If the state space comprises 3 states (s_1: initial state, s_2 and s_3), the transition function is defined on 48 couples (s, v_I); an enumerated description of this function is given in table 1, in which each line corresponds to a state s and each column to an inputs valuation v_I. Column 2, for instance, represents the inputs valuation such that c, o and r are False and v is True; this valuation can be represented either by the minterm $\overline{c} \cdot \overline{o} \cdot \overline{r} \cdot v$ or by the subset of I that contains the only variables which are True for this valuation $\{v\}$.

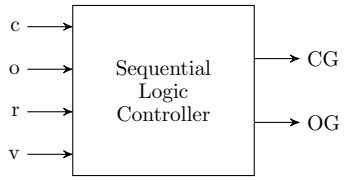


Fig. 3. Inputs/Outputs of the example

Each cell of the table contains the value of $\delta(s, v_I)$. A circled state name means that the same state is both source and target of the transition (self-loop structure). The output function is merely defined by:
$\lambda(s_1) = \overline{CG} \cdot \overline{OG}$; $\lambda(s_2) = CG \cdot \overline{OG}$; $\lambda(s_3) = \overline{CG} \cdot OG$;

The behavior given table 1 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. For example, when the active state is s_1, for the inputs valuations $\overline{c} \cdot \overline{o} \cdot \overline{r} \cdot v$, the system evolves towards state s_3. Since $\delta(s_3, \overline{c} \cdot \overline{o} \cdot \overline{r} \cdot v) = s_3$, no other evolution is possible without a new change of inputs valuation.

This tabular description cannot be used obviously for non-trivial examples. Hopefully, it can be replaced by an equivalent graphical description (Figure 4) in which a Boolean expression E_{ij} is associated to each transition from state s_i to state s_j. This expression represents the set of inputs valuations that cause this transition. For the example, E_{12} for instance is True only for the two inputs valuations $\overline{c} \cdot \overline{o} \cdot \overline{r} \cdot \overline{v}$ and $\overline{c} \cdot o \cdot \overline{r} \cdot \overline{v}$, then E_{12} represents the set of these two inputs valuations. This behavior can be obtained automatically from standardized language (Provost et al. (2009).

The determinism and completeness conditions can then be respectively reformulated as follows:

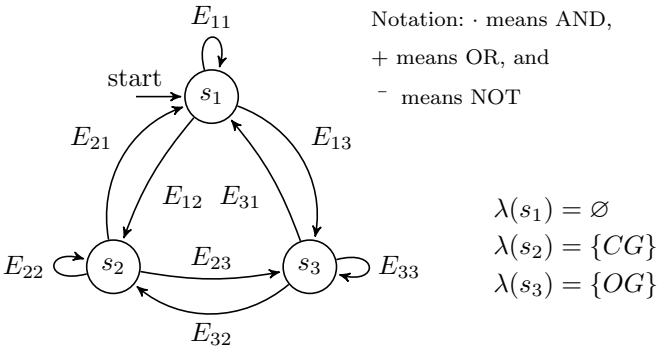
$$\forall (s_i, s_j, s_k) \in S^3, \ s_j \neq s_k \Rightarrow E_{ij} \cdot E_{ik} = Exp_{AlwaysFalse} \quad (7)$$

$$\forall s_i \in S, \ \sum_{s_j \in S} E_{ij} = Exp_{AlwaysTrue} \quad (8)$$

where $Exp_{AlwaysFalse}$ and $Exp_{AlwaysTrue}$ correspond to the Boolean expressions "always False" and "always True".

The condition on absence of transient evolution becomes:

$$\forall (s_i, s_j) \in S^2, \ E_{ij} \cdot \overline{E_{jj}} = Exp_{AlwaysFalse} \quad (9)$$



$E_{11} = o \cdot (c + r + v) + c \cdot \overline{r}$ $\quad$ $E_{21} = o \cdot (c + r + v) + c \cdot \overline{r}$
$E_{22} = \overline{c} \cdot \overline{r} \cdot \overline{v}$ $\quad$ $E_{23} = \overline{o} \cdot r + \overline{c} \cdot \overline{o} \cdot v$
$E_{33} = \overline{o}$ $\quad$ $E_{31} = o \cdot (c + r + v)$
$E_{12} = \overline{c} \cdot \overline{r} \cdot \overline{v}$ $\quad$ $E_{32} = \overline{c} \cdot o \cdot \overline{r} \cdot \overline{v}$
$E_{13} = \overline{o} \cdot (r + \overline{c} \cdot v)$

Fig. 4. Graphical representation of the behavior

3. SIC TEST SEQUENCE DEFINITION

3.1 Formal definition of test sequences

During test execution, a test sequence is seen as an ordered list of couples (inputs valuation,expected outputs valuation) which represents its external view:

$$\left[(v_I^0, v_O^0), (v_I^1, v_O^1), ..., (v_I^n, v_O^n)\right] \in (V_I \times V_O)^* \quad (10)$$

Conformance test implies that the expected outputs valuation is obtained from the behavior model. More precisely, this valuation is associated to the target state of the evolution that is provoked by the inputs valuation, when the model is in a given source state. To obtain this expected valuation, the source state (s_s) and target state (s_t) must compulsorily be known. Hence, an elementary conformance test step et is defined by the following 4-tuple:

$$et = (s_s, v_I, s_t, v_O) \in S \times V_I \times S \times V_O$$
$$\text{where } \begin{cases} s_t = \delta(s_s, v_I) \\ v_O = \lambda(s_t) = \lambda(\delta(s_s, v_I)) \end{cases} \quad (11)$$

Thus, a test sequence is an ordered list of elementary test steps and a consistent test sequence TS is a test sequence such as the source state of the k^{th} elementary test step is equal to the target state of the $(k-1)^{th}$ step.

$$TS = [(s^0, v_I^0, \delta(s^0, v_I^0), \lambda(\delta(s^0, v_I^0))), \cdots, (s^n, v_I^n, \delta(s^n, v_I^n), \lambda(\delta(s^n, v_I^n)))] \mid \forall k > 1, s^k = \delta(s^{k-1}, v_I^{k-1}) \quad (12)$$

(s, v_O): (States,Outputs valuations) \ v_I: Inputs valuations	$\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}$ or $\{\}$	$\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v$ or $\{v\}$	$\overline{c}\cdot\overline{o}\cdot r\cdot\overline{v}$ or $\{r\}$	$\overline{c}\cdot\overline{o}\cdot r\cdot v$ or $\{r,v\}$	$\overline{c}\cdot o\cdot\overline{r}\cdot\overline{v}$ or $\{o\}$	$\overline{c}\cdot o\cdot\overline{r}\cdot v$ or $\{o,v\}$	$\overline{c}\cdot o\cdot r\cdot\overline{v}$ or $\{o,r\}$	$\overline{c}\cdot o\cdot r\cdot v$ or $\{o,r,v\}$	$c\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}$ or $\{c\}$	$c\cdot\overline{o}\cdot\overline{r}\cdot v$ or $\{c,v\}$	$c\cdot\overline{o}\cdot r\cdot\overline{v}$ or $\{c,r\}$	$c\cdot\overline{o}\cdot r\cdot v$ or $\{c,r,v\}$	$c\cdot o\cdot\overline{r}\cdot\overline{v}$ or $\{c,o\}$	$c\cdot o\cdot\overline{r}\cdot v$ or $\{c,o,v\}$	$c\cdot o\cdot r\cdot\overline{v}$ or $\{c,o,r\}$	$c\cdot o\cdot r\cdot v$ or $\{c,o,r,v\}$
$(s_1, \overline{CG}\cdot\overline{OG})$	s_2	s_3	s_3	s_3	s_2	(s_1)	(s_1)	(s_1)	(s_1)	(s_1)	s_3	s_3	(s_1)	(s_1)	(s_1)	(s_1)
$(s_2, CG\cdot\overline{OG})$	(s_2)	s_3	s_3	s_3	(s_2)	s_1	s_1	s_1	s_1	s_1	s_3	s_3	s_1	s_1	s_1	s_1
$(s_3, \overline{CG}\cdot OG)$	(s_3)	(s_3)	(s_3)	(s_3)	s_2	s_1	s_1	s_1	(s_3)	(s_3)	(s_3)	(s_3)	s_1	s_1	s_1	s_1

Table 1. Example of behavior given in an extensional form

3.2 Properties of test sequences

A test sequence TS may be:

- initializable, i.e. the source state of the first test step is the initial state:

$$s^0 = s_{Init} \tag{13}$$

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

$$\forall (s, v_I) \in (S \times I),\ (s, v_I, \delta(s, v_I), \lambda(\delta(s, v_I))) \in TS \tag{14}$$

- based on a SIC inputs sequence.

To express formally this latter property, the SIC relation between two inputs valuations must be first defined. The definition below is based on the representation of an inputs valuation by the subset of I that contains the only variables which are True for this valuation. Thus, two inputs valuations v_I and v'_I satisfy a SIC relation if and only if[3]:

$$dim((v_I \backslash v'_I) \cup (v'_I \backslash v_I)) = 1 \tag{15}$$

For example, the inputs valuations 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 $dim((\{r,v\}\backslash\{c,r,v\}) \cup (\{c,r,v\}\backslash\{r,v\})) = dim(\varnothing \cup \{c\}) = 1$

In the remainder of this paper, this symmetrical relation is noted: $v_I\ R_{Gray}\ v'_I$. It can be noted that n SIC relations can be stated for each inputs valuation v_I of a SLC with n logic inputs.

Hence, a test sequence TS is based on a SIC inputs sequence and termed SIC test sequence if and only if:

$$\forall k > 1,\ v_I^k\ R_{Gray}\ v_I^{k-1} \tag{16}$$

3.3 Illustration on the example

In table 1, each cell may be associated to a test step whose source state, inputs valuation and target state are respectively given by the corresponding line, column and content of the cell. An initializable and exhaustive test sequence must start from a cell of the first line (initial state: s_1) and cover all cells of the table. A consistent test sequence corresponds to only horizontal and vertical cells changes according to the following rules:

- From the cells that contain a circled state name (the source and target states of the associated test step are identical), only horizontal changes are possible. The inputs valuations of two successive test steps can be different, but the source state of the second step must be identical to the target state of the first one.
- From the cells that contain a non-circled state name (the source and target states of the associated test step are different), only one vertical change is possible. This change leads to the cell that belongs to the line of the target state of the first test step.

It is easy to see that only the first case must be considered to determine whether the test sequence is a SIC test sequence, i.e. the inputs valuations v_I and v'_I of all couples of successive test steps satisfy $v_I\ R_{Gray}\ v'_I$. However, it is not always possible to build an exhaustive SIC test sequence from a behavioral model. For the example, the cell $(s_2, \overline{c}\cdot o\cdot r\cdot v)$ cannot be reached from cells $(s_2, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$ or $(s_2, \overline{c}\cdot o\cdot\overline{r}\cdot\overline{v})$ with only a SIC inputs sequence.

Then, SIC-testability of the behavioral model, ability of this model to be used to construct a SIC, initializable and exhaustive test sequence, must be checked before test sequence construction. This is the objective of the next section.

4. CHECKING SIC-TESTABILITY OF A SLC

This section aims to show that the SIC-testability of a SLC can be checked by a fixed-point calculation on elementary test steps. Before presenting this contribution, some details on conformance test execution must be reminded. An elementary test step is executed as follows:

- First, the test-bench sends to the controller under test the inputs valuation.
- Then, the test-bench waits that the controller computes its outputs; this waiting time is function of the technological features of the controller (filtering time, scanning cycle duration, ...) and will not be more discussed in this paper.
- Once the controller's outputs computed and stable, the test-bench compares the real outputs valuation to the expected one. The next step can then start.

[3] $dim(A)$ is the dimension of set A.
$v'_I \backslash v_I$ is the subset of I composed with elements of v_I which are not in v'_I.

(s, v_O): (States,Outputs valuations) \ v_I: Inputs valuations	$\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}$	$\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v$	$\overline{c}\cdot\overline{o}\cdot r\cdot\overline{v}$	$\overline{c}\cdot\overline{o}\cdot r\cdot v$	$\overline{c}\cdot o\cdot\overline{r}\cdot\overline{v}$	$\overline{c}\cdot o\cdot\overline{r}\cdot v$	$\overline{c}\cdot o\cdot r\cdot\overline{v}$	$\overline{c}\cdot o\cdot r\cdot v$	$c\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}$	$c\cdot\overline{o}\cdot\overline{r}\cdot v$	$c\cdot\overline{o}\cdot r\cdot\overline{v}$	$c\cdot\overline{o}\cdot r\cdot v$	$c\cdot o\cdot\overline{r}\cdot\overline{v}$	$c\cdot o\cdot\overline{r}\cdot v$	$c\cdot o\cdot r\cdot\overline{v}$	$c\cdot o\cdot r\cdot v$
$(s_1, \overline{CG}\cdot\overline{OG})$	$^1s_2$	$^1s_3$	$^1s_3$	$^1s_3$	$^1s_2$	$^0(s_1)$	$^0(s_1)$	$^0(s_1)$	$^0(s_1)$	$^0(s_1)$	$^1s_3$	$^1s_3$	$^0(s_1)$	$^0(s_1)$	$^0(s_1)$	$^0(s_1)$
$(s_2, CG\cdot\overline{OG})$	$^1(s_2)$	$^2s_3$	$^2s_3$	s_3	$^1(s_2)$	$^2s_1$	$^2s_1$	s_1	$^2s_1$	s_1	s_3	s_3	$^2s_1$	s_1	s_1	s_1
$(s_3, \overline{CG}\cdot OG)$	$^2(s_3)$	$^1(s_3)$	$^1(s_3)$	$^1(s_3)$	$^3s_2$	$^2s_1$	$^2s_1$	$^2s_1$	$^2(s_3)$	$^2(s_3)$	$^1(s_3)$	$^1(s_3)$	$^3s_1$	$^3s_1$	$^2s_1$	$^2s_1$

Table 2. Illustration of the steps of the fixed point calculation of the SIC region

It matters to underline that, during the waiting time of the elementary test step (s_s, v_I, s_t, v_O), the test step (s_t, v_I, s_t, v_O), which specifies that v_I does not modify the active state when this state is s_t, is also automatically executed.

4.1 Principle

The proposed method is based on the following two observations:

- An elementary test step (s_s, v_I, s_t, v_O) for a behavior $\mathcal{B}$ is SIC-testable if it can be included into an initializable SIC test sequence for this behavior. Owing to the test execution features, the elementary test step (s_t, v_I, s_t, v_O) is also SIC-testable.
- If the elementary step (s_t, v_I, s_t, v_O) is SIC-testable, it is always possible to add to the test sequence an elementary step $(s_t, v'_I, \delta(s_t, v'_I), \lambda(s_t, v'_I))$ where v'_I satisfies: $v'_I \; R_{Gray} \; v_I$.

On these bases, the set of elementary test steps which are SIC-testable can be obtain by the following fixed point calculation:

$$\begin{aligned} R_{SIC}(n+1) = R_{SIC}(n) \cup \\ \big\{(s_k, v_I^{k+1}, \delta(s_k, v_I^{k+1}), \lambda(\delta(s_k, v_I^{k+1})))\big\} \cup \\ \big\{(\delta(s_k, v_I^{k+1}), v_I^{k+1}, \delta(s_k, v_I^{k+1}), \lambda(\delta(s_k, v_I^{k+1})))\big\} \mid \\ \exists (s_k, v_I^k) \in R_{SIC}(n) \mid \begin{cases} \delta(s_k, v_I^k) = s_k \\ v_I^{k+1} \; R_{Gray} \; v_I^k \end{cases} \end{aligned} \tag{17}$$

At the end of this iterative calculation, the controller is SIC-testable if the final set contains all test steps that can be defined from its behavior description. Otherwise, this final set defines the SIC-testable part.

As a real logic controller can always be put in its initial state before being tested, it was chosen to start the fixed-point calculation with the following set:

$$\begin{aligned} R_{SIC}(0) = \big\{(s_{Init}, v_I^0, s_{Init}, \lambda(s_{Init})) \mid \\ v_I^0 \in V_I, \; \delta(s_{Init}, v_I^0) = s_{Init}\big\} \end{aligned} \tag{18}$$

4.2 Illustration on the example

Table 2 presents the results of this calculation for the example. The number k of the iteration during which the test step was found SIC-testable is at the top-left corner of each cell. For example, the step associated to the cell $(s_1, c\cdot\overline{o}\cdot\overline{r}\cdot v)$ is obtained at iteration 0 (initialization), because this step corresponds to a self-loop on s_1. The step associated to the cell $(s_1, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v)$ is obtained in step 1, as $c\cdot\overline{o}\cdot\overline{r}\cdot v \; R_{Gray} \; \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v$, and so on. Calculation stops at the third iteration, excluding the initialization. The final set contains only 40 test steps; the steps that do not belong to this set are represented by colored cells. Hence, the behavior of this controller is NOT SIC-testable; its SIC-testable part is given by the cells which are not colored.

4.3 Implementation for graphical descriptions of behavior

As already mentioned, the tabular representation is not suitable to specify non-trivial controllers. Hence, the fixed-point calculation has been transformed to deal with graphical descriptions of the behavior. The core idea of this transformation is to manipulate the inputs valuations through Boolean expressions and not explicitly. Algorithm 1 has been then developed to automate SIC-testability analysis.

This algorithm relies mainly on an operation noted $Expansion_{Gray}$ that is based on symbolic calculus and permits to expand a given set V_I^A of inputs valuations v_I with the inputs valuations v'_I which satisfy $v'_I \; R_{Gray} \; v_I$. The expanded set is noted V_I^B:

$$V_I^B = V_I^A \cup \{v'_I \mid \exists v_I \in V_I^A : \; v'_I \; R_{Gray} \; v_I\}, \tag{19}$$

Let us consider Exp_A the Boolean expression which represents the set V_I^A. The set V_I^B is described by the following Boolean expression:

$$\begin{aligned} Exp_B &= Expansion_{Gray}(Exp_A) \\ &= \sum_{i \in I} \left(Exp_{A_{|i \leftarrow False}} + Exp_{A_{|i \leftarrow True}}\right) \end{aligned} \tag{20}$$

The example below illustrates this operation.

$$\begin{aligned} V_I^A &= \{\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v \,,\, \overline{c}\cdot\overline{o}\cdot r\cdot\overline{v} \,,\, \overline{c}\cdot\overline{o}\cdot r\cdot v\} \\ Exp_A &= \overline{c}\cdot\overline{o}\cdot r + \overline{c}\cdot\overline{o}\cdot v \\ Exp_B &= (\overline{o}\cdot r + \overline{o}\cdot v) + (\overline{c}\cdot r + \overline{c}\cdot v) + (\overline{c}\cdot\overline{o}) + (\overline{c}\cdot\overline{o}) \\ &= \overline{c}\cdot\overline{o} + \overline{c}\cdot r + \overline{c}\cdot v + \overline{o}\cdot r + \overline{o}\cdot v \\ V_I^B &= \{\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v} \;,\; \overline{c}\cdot\overline{o}\cdot r\cdot\overline{v} \;,\; c\cdot\overline{o}\cdot r\cdot\overline{v} \;,\; \overline{c}\cdot o\cdot r\cdot\overline{v} \;, \\ &\quad \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v \;,\; c\cdot\overline{o}\cdot\overline{r}\cdot v \;,\; \overline{c}\cdot o\cdot\overline{r}\cdot v \;,\; \overline{c}\cdot\overline{o}\cdot r\cdot v \;, \\ &\quad c\cdot\overline{o}\cdot r\cdot v \;,\; \overline{c}\cdot o\cdot r\cdot v\} \end{aligned} \tag{21}$$

Algorithm 1 Calculation of the SIC-testable part of a behavior

Inputs: $\mathcal{B}$: $(S, s_{Init}, \delta, \lambda)$
Outputs: SIC-testable part of a behavior $\mathcal{B}$

```
/* Initialization step: */
for all s_i in S do
  T_SIC(s_i) := Exp_AlwaysFalse /* SIC-testable inputs valuations of s_I */
  Acc(s_i) := Exp_AlwaysFalse /* Inputs valuations of s_I accessible with a SIC sequence */
end for
T_SIC(s_Init) := E_11 /* As s_Init = s_1 */
Acc(s_Init) := E_11 /* As s_Init = s_1 */
/* Iterative construction: */
Improvements := True
while Improvements do
  Improvements := False
  for all s_i in S do
    New_SIC(s_i) := Expansion_Gray(Acc(s_i)) · not(T_SIC(s_i))
    if New_SIC(s_i) ≠ Exp_AlwaysFalse then
      Improvements := True
      T_SIC(s_i) := T_SIC(s_i) + New_SIC(s_i)
      for all evolution from s_i to s_j do
        new := New_SIC(s_i) · E_ij
        if new ≠ Exp_AlwaysFalse then
          Acc(s_j) := Acc(s_j) + new
        end if
      end for
    end if
  end for
end while
/* Display step: */
for s_i in S do
  TestablePart := T_SIC(s_i)
  print "SIC-Testable part of ",s_i,": ",TestablePart
end for
```

5. CONSTRUCTION OF THE SIC TEST SEQUENCE

This construction is based on a graph whose nodes represent all couples (s, v_I) that can be defined on the SIC-testable part. The arcs (directed edges) between the nodes are defined as follows:

- only one arc starts from a node that corresponds to a couple (s, v_I) such as $\delta(s, v_I) \neq s$; the target node of this arc is the node that corresponds to $(\delta(s, v_I), v_I)$;
- n arcs start from a node that corresponds to a couple (s, v_I) such as $\delta(s, v_I) = s$; the target nodes of these arcs correspond to couples (s, v'_I) such as v'_I satisfies $v'_I \; R_{Gray} \; v_I$.

The test sequence is then obtained by searching a pre-Hamiltonian path, i.e. a path that visits each node at least once. A SIC-test sequence for the SIC-testable part of the example is given table 3. The top and bottom lines are only given to relate this sequence to figure 4 and table 1. This test sequence contains 35 test steps and permits to test the 40 couples (s, v_I) of the SIC-testable part of the specification since some test steps permit to test both couples (s, v_I) and $(\delta(s, v_I), v_I)$. 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. For this example, the calculation of the SIC part uses symbolic computation and lasts less than 50 ms. The test sequence given table 3 is obtained in approximately 4 s, this computation lasts longer because it relies on a NP-hard optimization problem: the traveling salesman problem.

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

s_3	s_1	s_3	s_3	s_1	s_3	s_3	s_1	s_2	s_1	s_2	s_1	s_3	s_1	s_2	s_1	s_2	s_2
1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	1	1	1	1	1	0	1	1	1	1	0	0
1	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
s_1	s_3	s_3	s_1	s_3	s_3	s_1	s_2	s_1	s_2	s_1	s_3	s_1	s_2	s_1	s_2	s_2	s_1

Table 3. Test sequence for the SIC-testable part of the example

6. CONCLUSION

The main contributions of this work are the formal definition of SIC-testability and a formal method to obtain the SIC-testable part of a sequential logic controller. These results have been applied to build test sequences for conformance test of industrial PLCs (Programmable Logic Controllers) which are used for the control of critical systems.

Further work is aiming at assessing how the knowledge of the SIC and non-SIC-testable parts of a specification model impacts trustworthiness of test results, according to the test objective.

REFERENCES

da Silva Simão, A., Petrenko, A., and Yevtushenko, N. (2009). Generating reduced tests for FSMs with extra states. LNCS, 5826, 129–145.

Lee, D. and Yannakakis, M. (1996). Principles and methods of testing finite state machines - a survey. In Proceedings of the IEEE, volume 84, 1090–1123.

Provost, J., Roussel, J.M., and Faure, J.M. (2009). Test sequence construction from SFC specification. In Proceedings of 2nd IFAC Workshop on Dependable Control of Discrete Systems (DCDS'09). URL `http://hal.archives-ouvertes.fr/hal-00394454`.

Tretmans, J. (2008). Model based testing with labelled transition systems. LNCS, 4949, 1–38.

Virazel, A., David, R., Girard, P., Landrault, C., and Pravossoudovitch, S. (2001). Delay fault testing: Choosing between random sic and random mic test sequences. Journal of Electronic Testing: Theory and Applications, 17(3-4), 233–241.

von Bochmann, G. and Jourdan, G.V. (2009). Testing k-safe Petri nets. In TestCom/FATES - Testing of Software and Communication Systems, volume 5826 of LNCS, 33–48.

Yi, W., Xing-hua, F., and Dai-qiang, W. (2008). 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, 352–355.