

# Test sequence construction from SFC specification<sup>\*</sup>

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

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

**Abstract:** This paper focuses on conformance test of electronic programmable devices whose specification is given in Sequential Function Chart (SFC). More precisely, a method is proposed to obtain automatically, from this specification, one minimum length test sequence which permits the exhaustive test of the behavior of the device. This method takes advantage of previous results on construction of the state machine representation of a SFC and on test of Mealy machines; conversely, it extends the industrial use possibilities of this latter technique. The contribution is exemplified on a simple model. *Copyright ©2009 IFAC*

**Keywords:** conformance test, model-based test, SFC, Mealy machine, logic controllers.

## 1. INTRODUCTION

Critical systems are increasingly being controlled by electronic programmable devices, like ECUs (Electronic Control Units) for automotives or PLCs (Programmable Logic Controllers) for railway transport systems and power plants. To ensure systems' dependability, the conformance of these devices' behavior to control specifications must be tested. Roughly speaking, conformance test consists of (see Fig. 1):

- sending inputs sequence to the device;
- observing the response to this sequence;
- comparing the observed outputs sequence to the expected one.

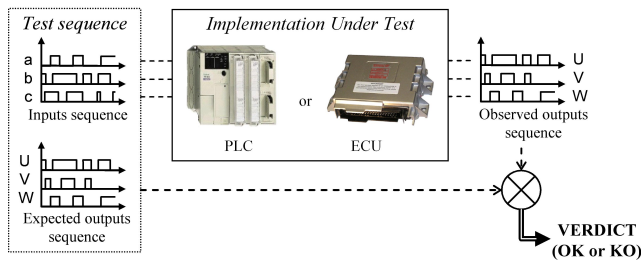


Fig. 1. Conformance test principle

Inputs and expected outputs sequences are termed a test sequence and are coming from the specification model. However, generating a test sequence manually is a very tedious and error-prone task. This explains why model-based test is a popular research issue that already yielded many significant results (see Broy et al. (2005)). The aim of this approach is to construct automatically a test sequence from specification in a well formalized language, e.g. Mealy machine or transition system (see Brinksma et al. (1990), Lee and Yannakakis (1996), Tretmans (2008)).

<sup>\*</sup> This work is funded by the French National Research Agency (TESTEC, Réf. TLOG 07-022)

However, to the best of our knowledge, none of these works has addressed the issue of test sequence construction from specification in a standardized industrial control language. The aim of this paper is to fill this gap when the control is specified in SFC (IEC60848 (2002)). More precisely, this paper presents a method to obtain, from a SFC which describes the expected behavior of a logic controller, one test sequence that permits to check whether a device that is assumed to implement this specification conforms to it (see Fig. 2). This test sequence must permit the exploration of the whole state space of the specification so as to obtain trustworthy test results, and be minimum length to reduce test duration.

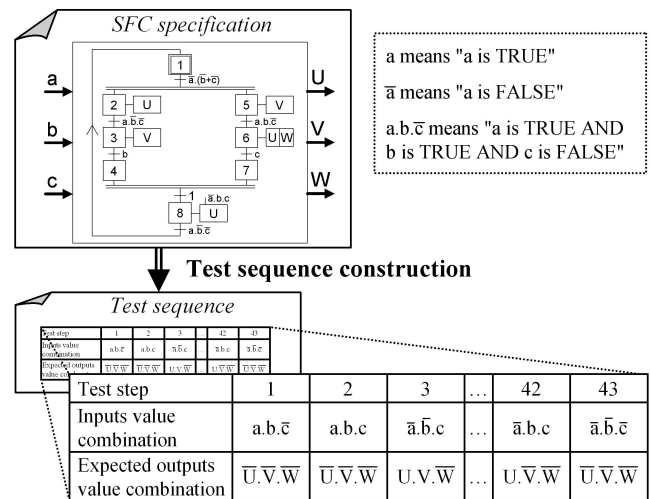


Fig. 2. Aim of the work

The different stages of this method are outlined in section 2, while the principles of conformance test of Mealy machines, a formal frame that underlies this research, are reminded in section 3. Section 4 is devoted to the description of the main contribution of this work: translation of the state machine equivalent to a SFC into a

Mealy machine. Finally, section 5 shows how a minimum length test sequence can be obtained from the result of this translation.

## 2. METHOD OVERVIEW

To provide trustworthy results, conformance test of devices which control highly critical systems must be:

- Black-box type. The internal structure of the device is unknown and its behavior can be obtained only by observing its responses to an inputs sequence.
- Non invasive. No probe or piece of code can be introduced within the device to get values of internal signals or variables during the test.
- Exhaustive. The whole state space of the specification model, a SFC in this work, must be explored. It will be assumed in the rest of this paper that the size of this state space is small enough to avoid combinatory explosion; scalability of the test sequence construction method will not be addressed. This assumption is quite reasonable when dealing with safety-related functions of highly critical systems.

Moreover, only non timed systems are considered in this paper. This implies that the SFC specification does not include any time-dependent transition condition or action.

Given these constraints and this limitation, a test sequence can be obtained by the following method which comprises three stages:

- (1) construction of the Reachable Situation Automaton (RSA) which represents the behavior of the initial SFC;
- (2) translation of this state machine with transition conditions into an event-based Mealy machine;
- (3) construction of the test sequence from this machine.

The aims of the first two stages are respectively to find all the underlying states of the SFC specification and all the event-based transitions between these states. It matters to remind indeed that:

- Several steps may be simultaneously active in a SFC; hence, at each date, the state of a SFC is a set of active steps variables, termed situation.
- A SFC transition can be fired for several inputs combinations; exhaustive test sequence construction requires to represent explicitly all these combinations.

At the end of these two stages, the whole state space of the specification is available. It is then possible, during stage 3, to build the test sequence.

A detailed presentation of stage 1 can be found in Roussel and Lesage (1996). The main results of this reference are recalled below:

- Any SFC which includes no time-dependent element (transition condition or action) can be translated into a state machine that describes all possible evolutions of this SFC. This state machine is called *RSA*. Each state of the RSA represents a situation of the initial SFC.
- A RSA includes only one active situation at each moment.

- Each situation of the RSA is reachable from the initial situation.
- All transition conditions of a RSA are Boolean expressions on the inputs of the initial SFC.
- The values of the outputs of the RSA only depend on the active situation.
- There is no transient evolution within a RSA (a formal statement of this feature will be given in section 4). If the initial SFC contains transient evolutions, they are replaced by non transient ones during RSA construction.

Last, to ensure non-invasive test, it will be assumed that a different set of actions is associated with each situation of the RSA. This permits to identify the active situation only from the values of the outputs.

Figure 3 depicts a simple case of RSA which will serve as example in the rest of this paper.

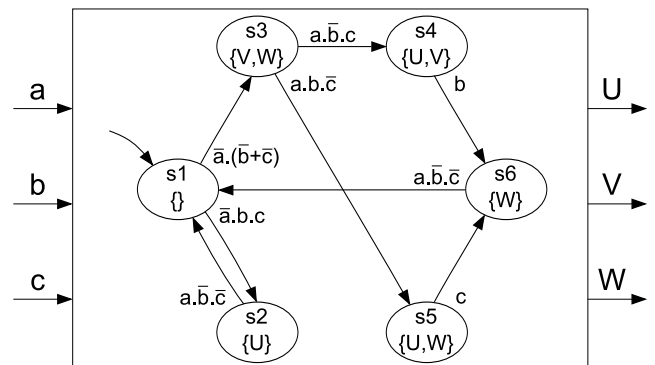


Fig. 3. Example of a Reachable Situation Automaton

Prior to detailing stage 2 of the test sequence construction method in section 4, it matters to remind the reader of the principle of conformance test of Mealy machines; this is the aim of the next section.

## 3. CONFORMANCE TEST OF MEALY MACHINES

Many researches about conformance test of Mealy machines have been achieved in the past. A good synthesis of these works is proposed in Lee and Yannakakis (1996). A brief description of their main results, based on the previous reference and on Broy et al. (2005) is given below.

Formally, a Mealy machine  $M$  is a 6-tuple  $(I_M, O_M, S_M, s_{initM}, \delta_M, \lambda_M)$  where:

- $I_M$  and  $O_M$  are nonempty sets of symbols, respectively, of inputs and outputs;
- $S_M$  is a nonempty set of states;
- $s_{initM} \in S_M$  is the initial state;
- $\delta_M: S_M \times I_M \rightarrow S_M$  is the transition function;
- $\lambda_M: S_M \times I_M \rightarrow O_M$  is the output function.

By definition, Mealy machines are deterministic since  $\delta_M$  and  $\lambda_M$  are defined by functions. It is generally assumed that these machines are complete, which means that the functions  $\delta_M$  and  $\lambda_M$  are defined for each 2-tuple  $(s, i) \in S_M \times I_M$ .

The figure 4 represents a Mealy machine with 4 states built on the input alphabet  $\{i, j\}$  and the output alphabet  $\{X, Y\}$ .

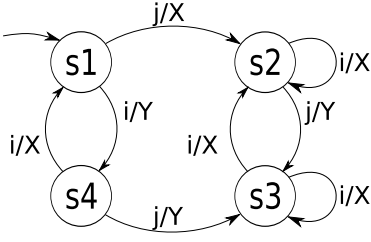


Fig. 4. Example of a Mealy machine

Within a Mealy machine, two states  $s_i$  and  $s_j$  are equivalent if for any inputs sequence  $(\sigma \in I_M^*)^1$ , the Mealy machine produces the same outputs sequence:

$$\forall \sigma \in I_M^* [\lambda_M(s_i, \sigma) = \lambda_M(s_j, \sigma)] \quad (1)$$

A Mealy machine is minimal if it does not include any pair of different equivalent states. Two machines  $M_1$  and  $M_2$  which have the same alphabet are said equivalent if for any state in  $M_1$  there is an equivalent state in  $M_2$  and vice versa.

Given these definitions, the problem of conformance test based on this model can be described as follows: Let  $S$  be a known machine (the specification) and  $I$  an unknown machine (the implementation under test) which can be only observed through its inputs and outputs, determine by a test that includes a finite sequence of inputs and expected outputs whether  $I$  is equivalent to  $S$  or not.

In order to solve this problem, it is generally assumed that the specification is minimal and strongly connected (each state is reachable from any other state). Then, the equivalence between an implementation  $I$  and a specification  $S$  consists in verifying that none of the following errors happens during the test of  $I$ :

- Output error:  $s$  being the active state, when the input  $i$  occurs,  $I$  produces the output  $o'$  instead of the expected output  $o$ .
- Transfer error:  $s$  being the active state, when the input  $i$  occurs, the transition labeled  $i/o$  is fired but the arrival state is  $s''$  instead of  $s'$ .

The test sequence is constructed from  $S$  and must permit to detect these two kinds of errors, for each state and each transition. Hence, each elementary test corresponds to a transition  $s \xrightarrow{i/o} s'$  of  $S$  and is defined as follows:

- (1) Go to  $s$  (synchronization).
- (2) Apply input  $i$  and check whether the emitted output is  $o$ .
- (3) Check whether the arrival state is  $s'$  (identification).

Then, the problems of synchronization and identification arise. To lessen or solve these problems, a Mealy machine can be endowed with two particular functions: *reset* and *status*. The *reset* function puts the Mealy machine in its initial state from the active state; the *status* function provides the value of the active state. The *reset* function permits to simplify synchronization because the path from the initial state to a given state  $s$ , source state of the transition which must be tested, is often shorter than the path from the active state to the state  $s$ . The *status*

function permits identification once a transition is fired and whatever the emitted output.

When the *status* function exists, a test sequence is usually obtained by the transition tour method (see Naito and Tsunoyama (1981)). The aim of this method is to find the shortest sequence of input symbols to cross at least once each arc of the graph corresponding to the Mealy machine. At the end of each stage of the test sequence, the *status* function is requested so as to determine the active state of the tested machine. Simplicity of implementation and efficiency are the main advantages of this method.

#### 4. TRANSLATION OF A REACHABLE SITUATION AUTOMATON INTO A MEALY MACHINE

This section is aiming to define the translation rules of a Reachable Situation Automaton (RSA) derived from a SFC into an event-based Mealy machine (stage 2 of the test sequence construction method proposed in section 2). The equivalence of behaviors of the resulting Mealy machine and the initial RSA is ensured by construction. It matters to highlight that RSA transitions are not labeled by events; only Boolean conditions are associated with these transitions. Conversely, a Mealy machine is an event-based model. Hence, the scientific issue to solve is translation of a condition-based finite state machine into an event-based one, without any semantics loss.

##### 4.1 Formal definition of Reachable Situation Automaton

A RSA can be formally represented by the following 6-tuple:  $(V_I, V_O, S_{RSA}, s_{initRSA}, T, A)$ , noted  $RSA$ , where:

- $V_I$  is a nonempty set of logical input variables, (Cardinality of  $V_I$ :  $|V_I| = n_{V_I}$ ).
- $V_O$  is a nonempty set of logical output variables, ( $|V_O| = n_{V_O}$ ).
- $S_{RSA}$  is a set of situations, ( $|S_{RSA}| = n_{S_{RSA}}$ ).
- $s_{initRSA} \in S_{RSA}$  is the initial situation.
- $T$  is a set of transitions  $t$ .
- $A$  is a set of actions  $a$ .

By construction,  $V_I$ ,  $V_O$  and  $S_{RSA}$  are distinct and nonempty sets.

A transition  $t$  of  $T$  is completely defined by the 3-tuple:  $t = (Up(t), Dw(t), C(t))$ , where:

- $Up(t)$  is the situation immediately before (Upstream) the transition  $t$ , ( $Up(t) \in S_{RSA}$ ).
- $Dw(t)$  is the situation immediately after (Downstream) the transition  $t$ , ( $Dw(t) \in S_{RSA}$ ).
- $C(t)$  is the transition condition associated to the transition  $t$ . A transition condition  $C(t)$  is a Boolean condition only composed of input variables in  $V_I$ .

An action  $a$  of  $A$  is completely defined by the 2-tuple:  $a = (s(a), o(a))$ , where:

- $s(a)$  is the situation which the action is associated with, ( $s(a) \in S_{RSA}$ ).
- $o(a)$  is the output which is set by the action, ( $o(a) \in V_O$ ).

According to these definitions, the RSA illustrated in Fig. 3 can be represented by the following 6-tuple:

<sup>1</sup>  $I_M^*$  represents  $I_M$  power any strictly positive integer.

$$\left\{ \begin{array}{l} V_I = \{a, b, c\} \\ V_O = \{U, V, W\} \\ S_{RSA} = \{s_1, s_2, s_3, s_4, s_5, s_6\} \\ s_{initRSA} = s_1 \\ T = \{(s_1, s_2, \bar{a}.b.c), (s_1, s_3, \bar{a}.\bar{(b+c)}), \\ (s_2, s_1, a.\bar{b}.\bar{c}), (s_3, s_4, a.\bar{b}.c), (s_3, s_5, a.b.\bar{c}), \\ (s_4, s_6, b), (s_5, s_6, c), (s_6, s_1, a.\bar{b}.\bar{c})\} \\ A = \{(s_2, U), (s_3, V), (s_3, W), (s_4, U), (s_4, V), \\ (s_5, U), (s_5, W), (s_6, W)\} \end{array} \right. \quad (2)$$

Moreover, the RSA which has to be translated will be assumed well structured, i.e.:

- When a transition is fired, the active situation changes (no self-loop):

$$\forall t \in T [Up(t) \neq Dw(t)] \quad (3)$$

- Transitions are not redundant:

$$\forall (t_i, t_j) \in T^2 [Up(t_i) = Up(t_j) \Rightarrow Dw(t_i) \neq Dw(t_j)] \quad (4)$$

- Actions are not redundant:

$$\forall (a_i, a_j) \in A^2 [s(a_i) = s(a_j) \Rightarrow o(a_i) \neq o(a_j)] \quad (5)$$

- Each situation of the RSA is reachable from the initial situation; there exists at least one sequence of transitions from the initial situation to any situation:

$$\forall s \in \{S_{RSA} - \{s_{initRSA}\}\} \exists (t_1..t_n) \in T^n (n \in \mathbb{N}^*) \left[ \begin{array}{l} Up(t_1) = s_{initRSA} \\ Dw(t_n) = s \\ \forall i \in [1, n-1], Dw(t_i) = Up(t_{i+1}) \end{array} \right] \quad (6)$$

Last, the three following statements can be written. The first two ones are true for any RSA derived from an initial SFC; the third one holds only if a test sequence for non-invasive test, i.e. such that the active situation can be known only from the values of the outputs, can be constructed from the Mealy machine translation of the RSA. Hence, this statement will be used to check whether the assumption on observability of situations through outputs is valid.

- The RSA includes only one active situation at each moment, i.e. the transitions which follow a situation are mutually exclusive:

$$\forall s \in S_{RSA}, \forall (t_i, t_j) \in T^2 [Up(t_i) = Up(t_j) = s \Rightarrow C(t_i) \cdot C(t_j) = 0] \quad (7)$$

- There is no transient evolution within the RSA:

$$\forall s \in S_{RSA}, \forall (t_i, t_j) \in T^2 [Up(t_i) = Dw(t_j) = s \Rightarrow C(t_i) \cdot C(t_j) = 0] \quad (8)$$

- A different set of actions is associated with each situation:

$$\forall (s_i, s_j) \in S_{RSA}^2 [O(s_i) \neq O(s_j)] \quad (9)$$

where  $O(s_i)$  is the subset of  $V_O$  that contains all outputs which are set when situation  $s_i$  is active.

$$O(s_i) = \{o(a) \mid \exists a \in A [s(a) = s_i]\} \quad (10)$$

#### 4.2 Mealy machine construction

A Mealy machine  $M: (I_M, O_M, S_M, s_{initM}, \delta_M, \lambda_M)$  can be constructed from any Reachable Situation Automaton

$RSA: (V_I, V_O, S_{RSA}, s_{initRSA}, T, A)$ . This Mealy machine behaves strictly as the RSA and is defined as follows:

- $I_M$ : input alphabet. This alphabet contains  $2^{n_{V_I}}$  elements. Each element  $i_i$  of this alphabet must represent a different combination of logical variables of  $V_I$ . Each element  $i_i$  is associated with a distinct minterm<sup>2</sup> built on the variables of  $V_I$  (see Table 1 for the studied example). Let  $m_I(i_i)$  be the minterm associated with the input event  $i_i$ .  $m_I(i_i)$  is built as follows:

$$\forall (i_i, i_j) \in I_M^2 [m_I(i_i) \cdot m_I(i_j) = 0] \quad (11)$$

Table 1. Association input event/minterm

Input event	$i_0$	$i_1$	$i_2$	$i_3$	$i_4$	...	$i_7$
Minterm	$\bar{a}.\bar{b}.\bar{c}$	$\bar{a}.\bar{b}.c$	$\bar{a}.b.\bar{c}$	$\bar{a}.b.c$	$a.\bar{b}.\bar{c}$	...	$a.b.c$

- $O_M$ : output alphabet. This alphabet contains  $2^{n_{V_O}}$  elements. Each element  $o_i$  of this alphabet must represent a different combination of logical variables of  $V_O$ . Each element  $o_i$  is associated with a distinct minterm built on the variables of  $V_O$  (see Table 2 for the studied example). Let  $m_O(o_i)$  be the minterm associated with the output event  $o_i$ .  $m_O(o_i)$  is built as follows:

$$\forall (o_i, o_j) \in O_M^2 [m_O(o_i) \cdot m_O(o_j) = 0] \quad (12)$$

Table 2. Association output event/minterm

Output event	$o_0$	...	$o_5$	$o_6$	$o_7$
Minterm	$\bar{U}.\bar{V}.\bar{W}$	...	$U.\bar{V}.W$	$U.V.\bar{W}$	$U.V.W$

- $S_M$ : set of states. A state of the Mealy machine  $M$  is associated to each situation of the Reachable Situation Automaton  $RSA$ . To make the translation easier to understand, the state of  $M$  and the associated situation of  $RSA$  will be denoted in the same way. Thus, the set  $S_M$  is the same as the set  $S_{RSA}$  of  $RSA$ .

$$S_M \equiv S_{RSA} \quad (13)$$

- $s_{initM}$ : initial state. This state is associated with the initial situation of  $RSA$ .

$$s_{initM} \equiv s_{initRSA} \quad (14)$$

- The transition function  $\delta_M$  and output function  $\lambda_M$  are defined on the basis sets  $V_I, V_O, S_{RSA}, T, A$  and  $I_M, O_M, S_M$ . The Mealy machine must be deterministic and completely specified, i.e.:

$$\forall (s, i) \in S_M \times I_M \left[ \begin{array}{l} \exists! \delta_M(s, i) \in S_M \\ \exists! \lambda_M(s, i) \in O_M \end{array} \right] \quad (15)$$

The transition function  $\delta_M$  of the machine  $M$  is defined as follows:

$$\forall s \in S_M, \forall i \in I_M \left[ \begin{array}{l} \text{if } \exists t \in T \left[ \begin{array}{l} Up(t) = s \\ C(t) \cdot m_I(i) = m_I(i) \end{array} \right] \\ \left[ \begin{array}{l} \delta_M(s, i) = Dw(t) \\ \text{else} \\ \delta_M(s, i) = s \end{array} \right] \end{array} \right] \quad (16)$$

The transition function  $\delta_M$  is completely specified since a value is defined for each 2-tuple  $(s, i) \in S_M \times I_M$ . This

<sup>2</sup> A minterm is a logical expression of  $n$  variables that uses only the logical conjunction operator and the complement operator.

<sup>3</sup>  $\exists!$ : There exists exactly one.

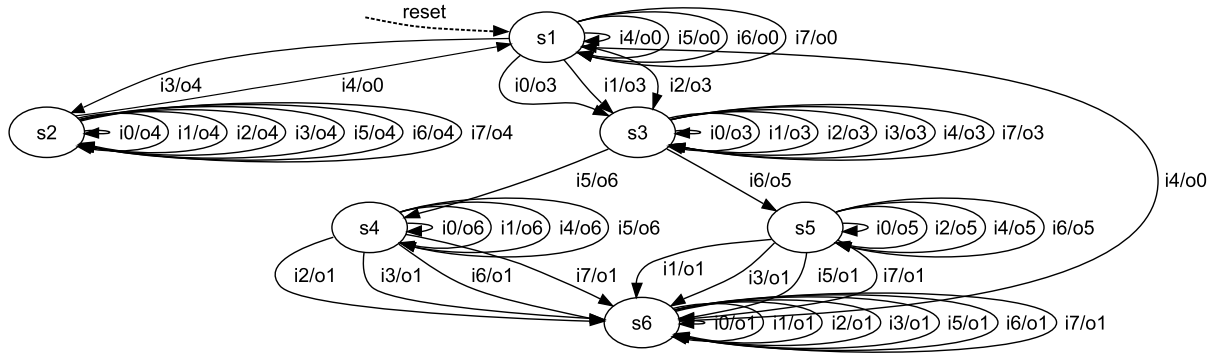


Fig. 5. Graphical representation of the Mealy machine corresponding to the RSA presented Fig. 3

value is unique because transitions which follow a situation  $s$  are mutually exclusive (see equation 7).

Construction of the output function  $\lambda_M$  relies on the property that outputs values depend only on the active situation of the RSA. Then, the output function  $\lambda_M$  is defined as follows:

$$\begin{aligned} & \forall s \in S_M, \forall i \in I_M [\lambda_M(s, i) = o_j] \\ & \text{where } \forall o_j \in O_M \\ & \left[ \begin{aligned} & \forall v \in O(\delta_M(s, i)) [m_O(o_j) \cdot v = m_O(o_j)] \\ & \forall v \in \{V_O - O(\delta_M(s, i))\} [m_O(o_j) \cdot \bar{v} = m_O(o_j)] \end{aligned} \right] \end{aligned} \quad (17)$$

The proposed definitions of  $\delta_M$  and  $\lambda_M$  ensure determinism of evolution and determinism of output events emission. The Mealy machine obtained is therefore deterministic and completely specified. Moreover, each state of this Mealy machine is reachable from the initial state since this constraint is true for the situations of the RSA (see equation 6).

The Mealy machine corresponding to RSA given in Fig. 3 is graphically represented in Fig. 5. Its mathematical definition is described by the following 6-tuple:

$$\begin{cases} I_M = \{i_0, i_1, i_2, i_3, i_4, i_5, i_6, i_7\} \\ O_M = \{o_0, o_1, o_2, o_3, o_4, o_5, o_6, o_7\} \\ S_M = \{s_1, s_2, s_3, s_4, s_5, s_6\} \\ s_{initM} = s_1 \\ \delta_M = S_M \times I_M \longrightarrow S_M \\ \lambda_M = S_M \times I_M \longrightarrow O_M \end{cases} \quad (18)$$

The tabular representation of transition function  $\delta_M$  and output function  $\lambda_M$  for this example is given in Table 3. In this table, the 2-tuple  $(\delta_M(s, i), \lambda_M(s, i))$  is associated to each 2-tuple  $(s, i)$ .

Table 3. Tabular representation of functions  $\delta_M(s, i)$  and  $\lambda_M(s, i)$  for the case study

s \ i	s <sub>1</sub>	s <sub>2</sub>	s <sub>3</sub>	s <sub>4</sub>	s <sub>5</sub>	s <sub>6</sub>
i <sub>0</sub>	s <sub>3</sub> ,o <sub>3</sub>	s <sub>2</sub> ,o <sub>4</sub>	s <sub>3</sub> ,o <sub>3</sub>	s <sub>4</sub> ,o <sub>6</sub>	s <sub>5</sub> ,o <sub>5</sub>	s <sub>6</sub> ,o <sub>1</sub>
i <sub>1</sub>	s <sub>3</sub> ,o <sub>3</sub>	s <sub>2</sub> ,o <sub>4</sub>	s <sub>3</sub> ,o <sub>3</sub>	s <sub>4</sub> ,o <sub>6</sub>	s <sub>6</sub> ,o <sub>1</sub>	s <sub>6</sub> ,o <sub>1</sub>
i <sub>2</sub>	s <sub>3</sub> ,o <sub>3</sub>	s <sub>2</sub> ,o <sub>4</sub>	s <sub>3</sub> ,o <sub>3</sub>	s <sub>6</sub> ,o <sub>1</sub>	s <sub>5</sub> ,o <sub>5</sub>	s <sub>6</sub> ,o <sub>1</sub>
i <sub>3</sub>	s <sub>2</sub> ,o <sub>4</sub>	s <sub>2</sub> ,o <sub>4</sub>	s <sub>3</sub> ,o <sub>3</sub>	s <sub>6</sub> ,o <sub>1</sub>	s <sub>6</sub> ,o <sub>1</sub>	s <sub>6</sub> ,o <sub>1</sub>
i <sub>4</sub>	s <sub>1</sub> ,o <sub>0</sub>	s <sub>1</sub> ,o <sub>0</sub>	s <sub>3</sub> ,o <sub>3</sub>	s <sub>4</sub> ,o <sub>6</sub>	s <sub>5</sub> ,o <sub>5</sub>	s <sub>1</sub> ,o <sub>0</sub>
i <sub>5</sub>	s <sub>1</sub> ,o <sub>0</sub>	s <sub>2</sub> ,o <sub>4</sub>	s <sub>4</sub> ,o <sub>6</sub>	s <sub>4</sub> ,o <sub>6</sub>	s <sub>6</sub> ,o <sub>1</sub>	s <sub>6</sub> ,o <sub>1</sub>
i <sub>6</sub>	s <sub>1</sub> ,o <sub>0</sub>	s <sub>2</sub> ,o <sub>4</sub>	s <sub>5</sub> ,o <sub>5</sub>	s <sub>6</sub> ,o <sub>1</sub>	s <sub>5</sub> ,o <sub>5</sub>	s <sub>6</sub> ,o <sub>1</sub>
i <sub>7</sub>	s <sub>1</sub> ,o <sub>0</sub>	s <sub>2</sub> ,o <sub>4</sub>	s <sub>3</sub> ,o <sub>3</sub>	s <sub>6</sub> ,o <sub>1</sub>	s <sub>6</sub> ,o <sub>1</sub>	s <sub>6</sub> ,o <sub>1</sub>

As shown in this representation of functions  $\delta_M(s, i)$  and  $\lambda_M(s, i)$ , each 2-tuple  $(s, i) \in S_M \times I_M$  verifies:

$$\begin{aligned} & \forall ((s, i), (s', i')) \in (S_M \times I_M)^2 \\ & [\delta_M(s, i) = \delta_M(s', i') \Rightarrow \lambda_M(s, i) = \lambda_M(s', i')] \end{aligned} \quad (19)$$

This result is a consequence of two features of the RSA (outputs values only depend on the active situation and a different set of actions is associated with each situation) and ensures minimality of the obtained Mealy machine.

The constructed Mealy machine contains as many states (or nodes in the graphical representation) as situations in the RSA. The number of transitions (or arcs in the graphical representation) only depends on the number of situations and input variables of the RSA.

$$\begin{cases} n_{states} = n_{RSA} \\ n_{transitions} = n_{RSA} \cdot 2^{n_{VI}} \end{cases} \quad (20)$$

For the case being considered, the obtained Mealy machine contains 6 states and 48 transitions. It must be noted that the number of transitions of the RSA has no influence on the size of the obtained event-based Mealy machine.

## 5. BUILDING THE TEST SEQUENCE

The Mealy machine which is constructed from a RSA represents all evolutions of the initial SFC. It is then possible to build from this machine a minimum length test sequence which can be used for conformance test of an EPD (Electronic Programmable Device), which is supposed to implement the SFC specification, by the transition tour method (see Naito and Tsunoyama (1981)). This method is a particular solution, for a graph which represents the structure of a Mealy machine, of a well-known problem in graph theory: the Chinese postman problem (see Mei-Ko (1962) and Edmonds and Johnson (1973)). The general formulation of this problem is the following: Find a minimum length closed walk that traverses each edge of the graph at least once.

A graph which describes the structure (states and transitions between states) of a Mealy machine is directed, but not weighted, which simplifies the optimization problem. In that case, it has been demonstrated (see previous references) that the optimal solution is the minimum length Eulerian cycle<sup>4</sup>, if the graph is Eulerian (every node has equal in and out degrees). If the graph is non Eulerian (at least one node has different in and out degrees), it must be transformed into an Eulerian one. This transformation consists in finding the shortest paths that connect in pairs

<sup>4</sup> An Eulerian cycle is a sequence whose start and end nodes are the same and that crosses only once each arc of the graph.

the nodes whose in and out degrees are different and doubling these paths as many times as necessary.

Table 4 provides the test sequence built for the graph obtained from the example of Figure 5. In this example, the in degree is greater than the out degree for the nodes which correspond to states  $s_3$  and  $s_6$  (the difference being 1 and 7 respectively), while the in degree is smaller than the out degree for the nodes which correspond to states  $s_1$ ,  $s_4$  and  $s_5$  (the difference being -2, -3 and -3 respectively). Hence, the structure of this Mealy machine is described by a non Eulerian graph. To obtain an Eulerian graph, arcs are to be added, as explained below:

- (1) The node that corresponds to state  $s_6$  is the node which has the greater difference between in and out degrees. The only solution to balance in an out degrees is to double 7 times the arc labeled  $i_4/o_0$  from  $s_6$  to  $s_1$ .
- (2) Then, the node which has the greater difference between in and out degrees is the node that corresponds to state  $s_1$ . A possible solution to balance in an out degrees is to double 5 times the arc labeled  $i_2/o_3$  from  $s_1$  to  $s_3$ .
- (3) Finally, nodes that correspond to states  $s_3$ ,  $s_4$  and  $s_5$  are balanced by doubling 3 times the arc labeled  $i_5/o_6$  from  $s_3$  to  $s_4$  and 3 times the arc labeled  $i_6/o_5$  from  $s_3$  to  $s_5$ .

Thus, 18 arcs are added. In practice, doubling an arc means crossing it several times. For example, the arcs labeled  $i_4/o_0$  from  $s_6$  to  $s_1$  and  $i_2/o_3$  from  $s_1$  to  $s_3$  are used respectively 8 and 5 times in the test sequence (Table 4). Then, the minimum length test sequence obtained starts from the initial state  $s_1$  and comprises 66 (48+18) elementary tests.

Table 4. Test sequence of the Mealy machine described in Fig. 5

Test step	1	2	3	4	5	6	7	8	9	10	
Input	$i_0$	$i_7$	$i_4$	$i_3$	$i_2$	$i_1$	$i_0$	$i_5$	$i_5$	$i_4$	
Output	$o_3$	$o_3$	$o_3$	$o_3$	$o_3$	$o_3$	$o_3$	$o_6$	$o_6$	$o_6$	
11	12	13	14	15	16	17	18	19	20	21	22
$i_1$	$i_0$	$i_3$	$i_7$	$i_6$	$i_5$	$i_3$	$i_2$	$i_1$	$i_0$	$i_4$	$i_7$
$o_6$	$o_6$	$o_1$	$o_1$	$o_1$	$o_1$	$o_1$	$o_1$	$o_1$	$o_1$	$o_0$	$o_0$
23	24	25	26	27	28	29	30	31	32	33	34
$i_6$	$i_5$	$i_4$	$i_2$	$i_5$	$i_7$	$i_4$	$i_2$	$i_5$	$i_6$	$i_4$	$i_2$
$o_0$	$o_0$	$o_0$	$o_3$	$o_6$	$o_1$	$o_0$	$o_3$	$o_6$	$o_1$	$o_0$	$o_3$
35	36	37	38	39	40	41	42	43	44	45	46
$i_6$	$i_6$	$i_4$	$i_2$	$i_0$	$i_3$	$i_4$	$i_2$	$i_6$	$i_7$	$i_4$	$i_2$
$o_5$	$o_5$	$o_5$	$o_5$	$o_5$	$o_1$	$o_0$	$o_3$	$o_5$	$o_1$	$o_0$	$o_3$
47	48	49	50	51	52	53	54	55	56	57	58
$i_6$	$i_5$	$i_4$	$i_2$	$i_6$	$i_1$	$i_4$	$i_1$	$i_5$	$i_2$	$i_4$	$i_3$
$o_5$	$o_1$	$o_0$	$o_3$	$o_5$	$o_1$	$o_0$	$o_3$	$o_6$	$o_1$	$o_0$	$o_4$
59	60	61	62	63	64	65	66				
$i_7$	$i_6$	$i_5$	$i_3$	$i_2$	$i_1$	$i_0$	$i_4$				
$o_4$	$o_4$	$o_4$	$o_4$	$o_4$	$o_4$	$o_4$	$o_0$				

This test sequence is used during the test as follows. The input signals that are sent to the EPD to test are built from the second line of table 4. The output signals issued from the tested EPD are compared to the expected output ones which are given by the third line of table 4. This comparison allows detection of both transfer and output errors, provided that a different set of outputs is associated

to each internal state of the EPD; it matters to underline that the *status* function is no more necessary to detect transfer errors if this condition is true.

## 6. CONCLUSION

This paper has shown how to construct, from a SFC specification, a minimum length test sequence to be used for exhaustive conformance test of EPDs. The main contribution of this work is the definition of formal rules to translate a SFC specification into a Mealy machine which represents all evolutions of this specification. The minimum length test sequence can be obtained then by the transition tour method.

On going works focus on three complementary issues so as to increase this contribution to dependability assessment of EPDs:

- introduction of the EPD scanning cycle model for test sequence generation;
- coupling to formal verification techniques so as to check the assumption on internal state observation through outputs values;
- extension of these results to more complex specifications, e.g. in the form of SFCs which include time dependent elements.

## REFERENCES

- Brinksma, E., Alderden, R., Langerak, R., van de Lage-maat, J., and Tretmans, J. (1990). A formal approach to conformance testing. In *J. de Meer, L. Mackert, and W. Effelsberg, editors, Second International Workshop on Protocol Test Systems*, 349–363.
- Broy, M., Jonsson, B., Katoen, J.P., Leucker, M., and Pretschner, A. (eds.) (2005). *Model-Based Testing of Reactive Systems, Advanced Lectures*, volume 3472 of *Lecture Notes in Computer Science*. Springer.
- Edmonds, J. and Johnson, E.L. (1973). Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5, 88–124.
- IEC60848 (2002). *GRAF CET specification language for sequential function charts*. 2. International Electrotechnical Commission.
- 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.
- Mei-Ko, K. (1962). Graphic programming using odd or even points. *Chinese Mathematics*, 1, 273–277.
- Naito, S. and Tsunoyama, M. (1981). Fault detection for sequential machines by transitions tours. In *Proceedings of the IEEE Fault Tolerant Computer Symposium*, 238–243.
- Roussel, J.M. and Lesage, J.J. (1996). Validation and verification of grafcet using state machine. In *Proceedings of IMACS-IEEE "CESA '96"*, 758–764. URL <http://hal.archives-ouvertes.fr/hal-00353188>.
- Tretmans, J. (2008). Model based testing with labelled transition systems. In R.M. Hierons, J.P. Bowen, and M. Harman (eds.), *Formal Methods and Testing*, volume 4949 of *Lecture Notes in Computer Science*, 1–38. Springer.