Translating Grafcet specifications into Mealy machines for conformance test purposes

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

Translating Grafcet specifications into Mealy machines for conformance test purposes

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

LURPA, ENS Cachan, 61, av du Président Wilson, Cachan, F-94230

Abstract

Conformance test is a black-box test technique aiming at checking whether an implementation conforms to its specification. Numerous results have been already obtained in this field for specifications expressed in a formal language. However, these results cannot be applied for conformance test of industrial logic controllers whose specifications are given in standardized specification languages. To contribute to solve this issue, this paper proposes a method to obtain, from a Grafcet specification, an equivalent Mealy machine, without semantics loss. This method permits to describe explicitly and formally all the states and transitions that are implicitly represented in a Grafcet model.

Keywords: Conformance test, Model-based test, Grafcet, Mealy machine, Logic controllers

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. Conformance test is a black-box test and 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 a test verdict (the implemented controller is conform or not). The set of the input sequence and expected output sequence is termed test sequence or test case.

Numerous theoretical results 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, more recently, a particular class of

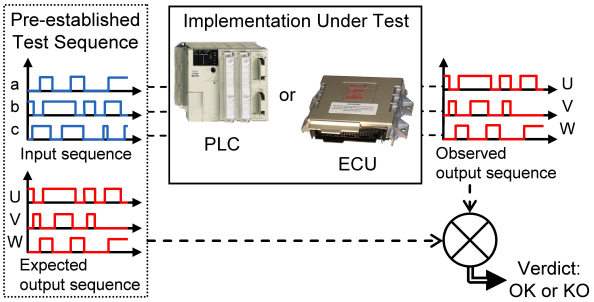


Figure 1: Conformance test principle

Petri net (von Bochmann and Jourdan (2009)). Generally speaking, these results provide a way to build automatically the test sequence from the formal specification model and to deliver a verdict from the observed output sequence.

In industrial practice, the specification of the behavior of logic controllers is not given in such formal models, however, but in tailor-made, (officially or de facto) standardized specification languages, like Grafcet or statecharts. Test cases are then built manually, what is a very tedious, time-consuming and error-prone task. To take benefit of previous works on conformance test based on formal models, it matters to endow the specification languages that are used in industry with a formal semantics and to develop translation methods of models

*Corresponding author. Fax: +33 147402220

Email address: julien.provost@lurpa.ens-cachan.fr, jean-marc.roussel@lurpa.ens-cachan.fr, jean-marc.faure@lurpa.ens-cachan.fr (Julien Provost, Jean-Marc Roussel, Jean-Marc Faure)

in these languages into formal ones. Several results on model-based conformance test from UML state-charts have been already published (Massink et al. (2006) for instance) but, as far as we know, the issue of conformance test when the specification is given in the form of a Grafcet, a powerful specification language for logic controllers, has never been addressed. The aim of this paper is to fill this gap.

More precisely, this paper proposes a method to translate a Grafcet specification model into an equivalent Mealy machine, without semantics loss (Figure 2). Mealy machines have been chosen as the formal target model because conformance test of Mealy machines is a mature technique that previously yielded numerous sound results, as surveyed in Lee and Yannakakis (1996). However, this choice implies that only non-timed systems are considered; then, the Grafcet specification model shall not contain any time-dependent element. This limitation is not too strong because the first concern of engineers during conformance test is functional correctness; only the correctness of the non-timed behavior of the controller with regard to the specification is checked. Conformance test for checking time correctness is a second concern, once functional correctness is ensured.

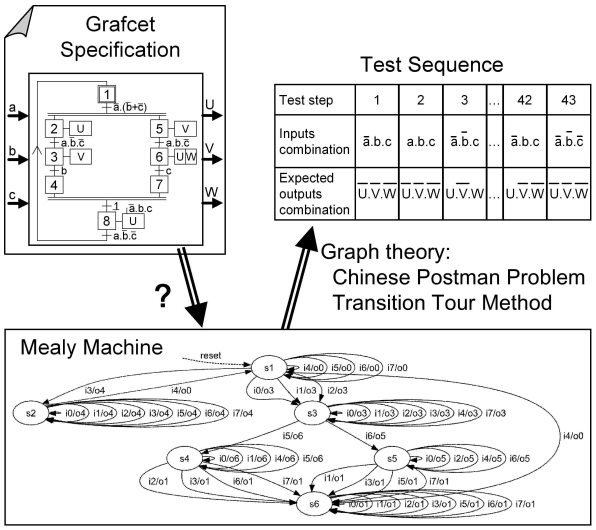


Figure 2: Objective of the work

Conformance test is a black-box test: the implementation is seen as a black-box with inputs/outputs. In the case of a logic controller, this means that its internal structure is unknown and its behavior can only be determined by observing its response to an input sequence. Moreover, to provide reliable results for controllers of highly critical systems, this test must be:

- Non-invasive. No probe or piece of code can be introduced within the controller. It is therefore impossible to obtain the values of its internal variables.
- Exhaustive. The whole state space of the specification model, a Grafcet model in this work, must be explored. In the rest of this paper, it will be supposed that the size of this state space is small enough to avoid combinatorial explosion. This assumption is quite reasonable for safety/security functions of critical systems. Indeed, since these functions must be very reactive (the response time to any change of their inputs must be very short), they do not perform complex treatments and the state space of the specification of such a function is tractable. For this reason, scalability of the test method will be not more addressed in what follows.

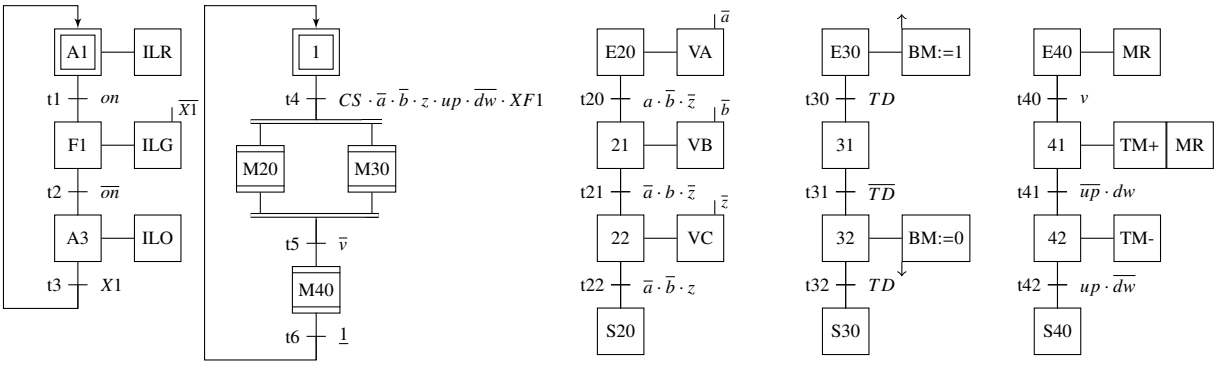
The model obtained by the translation method shall permit to satisfy these two test constraints.

The outline of the paper is the following. The background of this work - Grafcet syntax and standardized evolution rules as well as conformance test of Mealy machines - is reminded in the next section. An overview of the translation method is given in section 3. The two phases of this method are then detailed and illustrated on a small but non-trivial example, respectively in section 4 and 5. Then, section 6 focuses on test sequence generation from the final formal model, while perspectives for future works to extend this contribution are given in the conclusion.

2. Background

2.1. Grafcet specification language

Grafcet is a standardized graphical specification language (IEC 60848 (2002)) to describe the behavior of logic sequential systems. This language is widely used in several industrial domains, like railway transport, electrical power production, manufacturing industry, environment, to specify the expected behavior of a logic control system which is connected to a physical system (plant) that sends logic signals to the control system and receives the logic signals which are generated in response. Grafcet was first standardized in France at the beginning of the 1980s, and at the international level in 1988. Since this date, several extensions have been proposed to enhance the modeling possibilities; they are included in the last version of the



Notation: · means AND, + means OR, $\bar{}$ means NOT and $\underline{1}$ means "always True". For example, $\overline{up} \cdot dw$ means "up is False AND dw is True".

Inputs			
CS	Cycle Start	up	Mixer up
TD	Transit Detector	dw	Mixer down
a	Fluid weight A reached	on	Production is on
b	Fluid weight A + B reached	v	Viscosity reached
z	Empty weighing unit		

Outputs			
BM	Belt Motor	VB	Opening Valve B
MR	Mixer Rotation motor	VC	Opening Valve C
TM+	Tipping Motor (down)	ILG	Indicator Light Green
TM-	Tipping Motor (up)	ILO	Indicator Light Orange
VA	Opening Valve A	ILR	Indicator Light Red

Figure 3: Grafcet specification used to illustrate the proposed approach

standard (IEC 60848 (2002)). A good scientific presentation of the main features of the previous and current versions of the Grafcet standard can be found respectively in David (1996) and Guéguen and Bouteille (2001). Last, the reader is warned that the specification language described in the IEC 60848 standard differs from the SFC (Sequential Function Chart) proposed by the IEC 61131-3 standard (IEC 61131-3 (2003)), even if both are often named SFC in English and if models in these two languages may look similar; the differences stand both in syntax and semantics. The main differences between those two languages will be discussed in subsection 'Differences between Grafcet and SFC'. To avoid misunderstandings, only the term Grafcet will be kept in the sequel of this paper for the specification language.

Grafcet has been developed from the results of the Petri nets community and in particular from those on Interpreted Petri Nets. A specific syntax and semantics have been defined however, to take into account the specific needs of engineers when specifying complex sequential systems. The key features of Grafcet syntax and semantics are briefly recalled as follows.

Grafcet syntax

A Grafcet model describes the expected behavior of a logic controller which receives logic input signals and generates logic output signals; then, the input and output variables of a Grafcet are both logic variables. A Grafcet (Figure 3) comprises steps, graphically represented by squares, and transitions, represented by horizontal lines; a step can only be linked to transitions and a transition only linked to steps. The links from steps to transitions and from transitions to steps are oriented links. The default orientation is from top to bottom and it is not necessary in this case to put an arrow on the link. An arrow must be put on a link if this link goes from bottom to top or may be put on any link to ease understanding. A step defines a partial state of the system and can be active or inactive; hence, a Boolean variable, named step activity variable can be defined for each step. Actions may be associated to a step; an action associated to a step is performed only when this step is active and then acts upon an output variable. A transition condition must be associated to each transition; this condition is a Boolean expression which may include input variables, steps activity variables and conditions on time. As only non-timed systems are considered in this work, only the Grafcets whose transition conditions are built from input variables and steps activity variables are dealt with.

Moreover, macro-steps may be introduced in a Grafcet, to ease modeling. A macro-step, represented graphically by a square with double-lines on top and at the bottom, is a synthetic view of a part of the specification. The detailed description of this part is termed macro-step expansion chart and is a set of connected steps and transitions that starts and ends by only one step, called macro-step expansion input and output steps. Then, a Grafcet model may be composed of several charts: classical charts, that include (normal or macro) steps and transitions, and macro-step expansion charts.

Figure 3 depicts a Grafcet that comprises a part of an example given in the standard. This model is composed of two classical charts (on the left side) and three macro-step expansions (on the right side); these latter charts are the expansions of macro-steps 'M20', 'M30' and 'M40'.

Evolution rules

The detailed behavior of any Grafcet model can be obtained by applying five evolution rules that can be stated as follows:

1. At the initial time, all the initial steps, defined by the model designer and double-squared, are active; all the other steps are inactive.

2. A transition is enabled when all the steps that immediately precede this transition (upstream steps of the transition) are active. A transition is fireable when it is enabled and when the associated transition condition is true. A fireable transition must be immediately fired.

3. Firing a transition provokes simultaneously the activation of all the immediately succeeding steps and the deactivation of all the immediately preceding steps.

4. When several transitions are simultaneously fireable, they are simultaneously fired.

5. When a step shall be both activated and deactivated, by applying the above previous evolution rules, it is activated if it was inactive, or remains active if it was previously active.

This textual description of the evolution rules comes from the standard and is obviously not sufficient to translate a Grafcet into a formal model; a formal definition will be given in section 4.1. However, these rules show that the global state of a Grafcet, called situation, is defined by the set of all the simultaneously active steps; the initial situation of the model of figure 3, for instance, is {A1,1}. An evolution from the current situation to a new one corresponds to the firing of simultaneously fireable transitions, according to rules 2 (fireable transitions are compulsory fired) and 4 (simultaneously fireable transitions are simultaneously fired). This new situation may be *transient* or *stable*; a situation is transient if at least one transition of the Grafcet can be fired from this situation without change of the inputs values, and stable if no enabled transition is fireable from this situation for the current values of inputs. Then, the state of a Grafcet evolves from stable situation to stable situation, possibly by crossing transient situations; an evolution between two stable situations corresponds, in the Grafcet, to a sequence (may be reduced to one) of firings of sets of fireable transitions and is instantaneous. Examples of evolutions will be given in section 4, once the formal semantics of Grafcet defined.

Actions

Two kinds of actions can be used in a Grafcet to specify the values of outputs: continuous actions and stored actions. An action is graphically represented by a rectangle linked to the step symbol. A continuous action specifies the current value of an output according to the current values of steps activity variables and inputs. For figure 3, for instance, output 'MR' is true if and only if step 'E40' or step '41' is active; output 'VA' is true if and only if step 'E20' is active and input 'a' is false. A stored action describes how an output variable is allocated to a Boolean value, according to an allocation rule. A rising arrow associated to the action symbol means that the output variable is allocated when the step becomes active. On the other hand, a falling arrow means that the output variable is allocated when the step is deactivated. For the example, output 'BM' is allocated to true (Set) when step 'E30' becomes active and allocated to false (Reset) when step '32' is deactivated. It matters to underline that continuous actions are executed only if the current situation is stable whereas stored actions are executed whatever the situation.

This set of evolution rules and actions definitions ensures that Grafcet models are deterministic, what is mandatory for controllers' specifications.

Differences between Grafcet and SFC

This comparison is based on the two normative texts that define Grafcet and SFC, respectively IEC 60848 (2002) and IEC 61131-3 (2003). In what follows, the extracts from these documents will be written in italics.

Grafcet is a *specification language for the functional description of the behavior of the sequential part of a*

control system. On the opposite, SFC is an element to structure the internal organization of a program organization unit (derived from Grafcet) and written with one of the four IEC 61131-3 languages (LD, ST, IL, FBD). Then, a Grafcet model is used to specify a behavior (external view of the control system), while an SFC model describes (part of) the structure of a software running on a programmable controller that implements this behavior (internal view of the control system).

The semantics of Grafcet and SFC are different. For instance, in case of selection of sequence, *exclusive activation of a selected sequence is not guaranteed from the structure*, in Grafcet; *the designer should ensure that [...] the transition conditions are mutually exclusive*. On the contrary in SFC, *any sequence selection is exclusive, [...] it cannot have crossing simultaneous transitions in a sequence selection; to do this, the user can define priorities between branches at the divergence of sequence selection*.

But the main semantic difference is that the evolutions of a Grafcet model are caused by the changes of its inputs values whereas the evolutions of an SFC model are controlled by the input scanning cycle of the controller that executes this model. When the value of an input of a Grafcet model changes, this model evolves to a new stable situation; this evolution is instantaneous to avoid inputs values changes be missed, even if transient situations are crossed. On the opposite, in an SFC model, *the clearing time*[1] *of a transition may theoretically be considered as short as one may wish, but it can never be zero*; in practice, this time is equal to the input scanning cycle time of the controller on which the SFC runs. The consequence is that only one set of simultaneous fireable transitions (may be reduced to one transition) is fired at every scanning cycle. Hence, the concepts of transient and stable situations do not make sense in SFC; a situation lasts at least one cycle time.

To conclude and focus on the context of this work, in conformance test (validation of an implementation with respect to a specification), Grafcet is to be used to describe the specification; SFC is only a possible solution to implement this specification. From a given Grafcet, a test sequence will be constructed for testing an implementation which may be a PLC programmed in SFC (and, often, other IEC 61131-3 languages) or an ECU programmed in C, according to the implementation choices. The same Grafcet model can be the reference for different kinds of implementation.

[1] clearing time = firing time

2.2. Mealy machines testing

Many researches about conformance test of Mealy machines have been achieved in the past. A good synthesis of these works is proposed by 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.
- δ_M: $S_M \times I_M \longrightarrow S_M$ is the transition function.
- λ_M: $S_M \times I_M \longrightarrow O_M$ is the output function.

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

Within a Mealy machine, two states s_i and s_j are equivalent if for any inputs sequence[2] ($\sigma \in {I_M}^*$), the Mealy machine produces the same outputs sequence from s_i or s_j:

$$\forall \sigma \in {I_M}^* \left[\lambda_M(s_i, \sigma) = \lambda_M(s_j, \sigma)\right] \qquad (1)$$

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 Mealy machines 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 (there is no equivalent states) 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 (Figure 4):

[2] ${I_M}^*$ represents I_M power any strictly positive integer.

- 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).

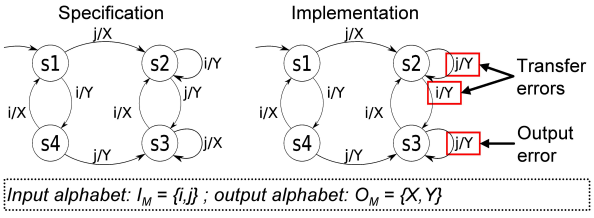


Figure 4: Example of transfer and output errors in a Mealy machine

3. Method overview

The translation method of a Grafcet into an equivalent Mealy machine, without semantics loss, comprises two phases (Figure 5):

- Construction of the automaton, termed stable location automaton (SLA), that represents formally all the stable states of the logic system described by the Grafcet as well as all the evolutions between these states.
- Translation of this automaton into an equivalent Mealy machine.

The state of the logic system that underlies a Grafcet model, called *location*, is featured by the set of simultaneously active steps (situation) and the set of emitted outputs; it is reminded indeed that different sets of outputs may be emitted for the same situation, when the

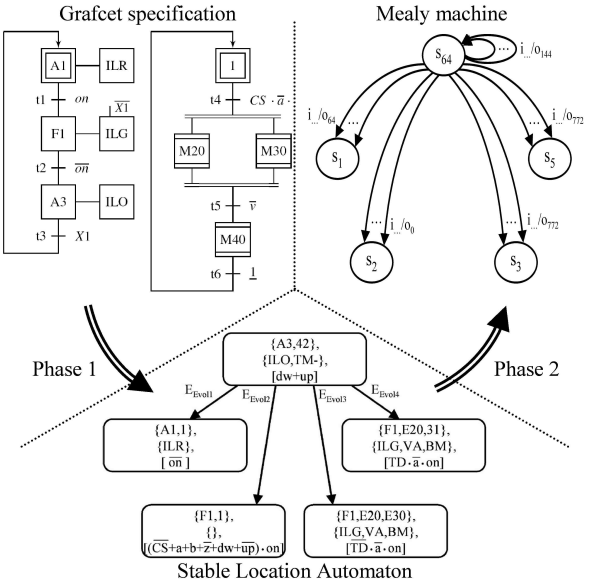


Figure 5: Method overview

Grafcet contains stored actions or continuous actions that depend on inputs values. As a situation may be transient or stable, a location may be transient or stable, too. Continuous actions are executed only when the situation of the Grafcet is stable; hence, the outputs that are controlled by continuous actions are emitted only for stable locations. To observe all outputs, whatever the kind (continuous/stored) of action they are controlled by, only the stable locations are to be kept. A stability condition, Boolean expression on the inputs values, is associated to each stable location; when a location is active[3] and its stability condition true, the SLA remains in this location, otherwise it evolves to a new location. Each evolution from a stable location to another stable location is caused by a change of the inputs values; an evolution condition, Boolean expression on the inputs values, will then be associated to each evolution. All these definitions will be formalized in section 4; a consistency rule on stability and evolution conditions will then be stated.

Once the SLA built, the aim of the second phase is to define the transition and output functions of the equivalent Mealy machine from the evolution condition, as formally detailed in section 5.

[3] By definition, only one location is active in an SLA at any moment.

4. Construction of the Stable Location Automaton of a given Grafcet model

To construct automatically the SLA of a given Grafcet, both semantics of Grafcet and SLA must be first defined; this is the objective of sections 4.1 and 4.2 respectively. It must be pointed out that formalization of Grafcet has been already addressed in Lhoste et al. (1993) and Bierel et al. (1997), by using static meta-modeling techniques and an algorithmic interpretation of the evolution rules, on the basis of the previous version of the standard which was issued in 1988. This work considers the 2002 version and is based on an algebraic approach to increase genericity. Equations to determine the locations and evolutions of the SLA are then stated in section 4.3.

4.1. Formal definition of a Grafcet model

Formally, a Grafcet g is a 4-tuple $(I_G, O_G, C_G, S_{InitG})$ where:

- I_G is the nonempty set of logic inputs, (Cardinality of I_G: $|I_G|$).
- O_G is the nonempty set of logic outputs.
- C_G is the set of Grafcet charts.
- S_{InitG} is the set of initial steps.

The charts set C_G is partitioned into the set C_C of classical charts and the set C_E of macro-steps expansions charts.

$$\begin{cases} C_C \cup C_E = C_G \\ C_C \cap C_E = \varnothing \end{cases} \tag{2}$$

A classical chart $c \in C_C$ is defined by a 3-tuple (S, T, A) where:

- S is the nonempty set of steps s of c.
- T is the set of transitions t of c.
- A is the set of actions a of c.

A macro-step expansion chart $c \in C_E$ is defined by a 5-tuple $(m, s_I, s_O, S_{oth}, T, A)$ where:

- m is the macro-step name.
- s_I is the input step of the expansion.
- s_O is the output step of the expansion.
- S_{oth} is the set of the other steps s of the expansion.
- T is the set of transitions t of the expansion.
- A is the set of actions a associated to the steps of the expansion.

For all macro-step expansion chart $c \in C_E$, $S(c)$ is the set of all steps of c.

$$S(c) = \{s_I(c), s_O(c)\} \cup S_{oth}(c) \tag{3}$$

The set of all steps s of the Grafcet will be denoted S_G.

$$S_G = \bigcup_{c \in C_G} S(c) \tag{4}$$

A step activity variable $X(s)$ is associated to each step s. The set of initial steps S_{InitG} is a subset of S_G.

A transition $t \in T$ of a given chart $c \in C_G$ is defined by a 3-tuple $(S_U, S_D, E_{Cond(I_G,S_G)})$ where:

- S_U is the set of the immediately upstream steps of the transition, $S_U \subset S(c)$.
- S_D is the set of the immediately downstream steps of the transition, $S_D \subset S(c)$.
- $E_{Cond(I_G,S_G)}$ is the transition condition, Boolean expression on inputs and steps activity variables.

The set of all transitions t of the Grafcet will be denoted T_G.

$$T_G = \bigcup_{c \in C_G} T(c) \tag{5}$$

The set of actions A is partitioned into the set A_S of stored actions and the set A_C of continuous actions. Similarly, the outputs set O_G is partitioned into the set O_S of stored outputs (controlled by stored actions) and the set O_C of continuous outputs (controlled by continuous actions).

$$\begin{cases} A_S \cup A_C = A \\ A_S \cap A_C = \varnothing \end{cases} \tag{6}$$

$$\begin{cases} O_S \cup O_C = O_G \\ O_S \cap O_C = \varnothing \end{cases} \tag{7}$$

A continuous action $a_c \in A_C$ of a given chart c is defined by a 3-tuple $(s, o, E_{Cond(I_G,S_G)})$ where:

- s is the step which the action is associated with, $s \in S(c)$.
- o is the output which is assigned by the action, $o \in O_G$.
- $E_{Cond(I_G,S_G)}$ the continuous action condition, Boolean expression on inputs and steps activity variables.

The value of each continuous output is then defined by a Boolean expression on inputs and steps activity variables, as follows:

$$E_{Emit(I_G,S_G)}(o) = \sum_{\substack{a\in A_C \\ o(a)=o}} \left(X(s(a)) \cdot E_{Cond(I_G,S_G)}(a)\right) \quad (8)$$

A stored action $a_s \in A_S$ of a given chart c is defined by a 4-tuple (s,o,op,$inst$) where:

- s is the step which the action is associated with, $s \in S(c)$.
- o is the output which is allocated by the action, $o \in O_G$.
- op is the kind of allocation, $op \in \{Set, Reset\}$.
- $inst$ is the instant when the allocation is done, $inst \in \{Act, Deact\}$, where Act is the step activation instant and $Deact$ the step deactivation instant.

Then, the values of stored outputs are computed dynamically during the construction of the SLA (see subsection 4.3).

4.2. Formal definition of the Stable Location Automaton (SLA) of a given Grafcet model

Formally, a stable location automaton SLA is a 5-tuple ($I_{SLA}, O_{SLA}, L, l_{Init}, Evol$) where:

- I_{SLA} is the set of inputs of the Grafcet model g: $I_{SLA} = I_G(g)$.
- O_{SLA} is the set of outputs of g: $O_{SLA} = O_G(g)$.
- L is a set of stable locations l.
- l_{Init} is the initial location, $l_{Init} \in L$.
- $Evol$ is a set of evolutions e.

It is reminded that a stable location is characterized by a set of simultaneously active steps, a set of emitted outputs and a stability condition, Boolean expression on the inputs values. Then, a stable location is defined by the 3-tuple: ($S_{Act}, O_{Em}, E_{Stab(I_G)}$) where:

- S_{Act} is a subset of steps of g, ($S_{Act} \subset S_G(g)$).
- O_{Em} is a subset of emitted outputs of g, ($O_{Em} \subset O_G(g)$).
- $E_{Stab(I_G)}$ is a Boolean expression on inputs of g. This expression is true only for the combinations of inputs values for which the location l is stable.

An evolution e of $Evol$ is defined to represent an evolution between two stable locations. Each one of these evolutions can be formally defined by the 3-tuple: ($l_U, l_D, E_{Evol(I_G)}$), where:

- l_U is the upstream location, $l_U \in L$.
- l_D is the downstream location, $l_D \in L$.
- $E_{Evol(I_G)}$ is the evolution condition, Boolean expression on inputs of g. This expression is true only for the combinations of inputs values for which the SLA evolves from location l_U to location l_D.

A SLA is well-defined if the following seven properties are satisfied:

- Distinguishability of locations:

$$\begin{array}{l} \forall (l_1, l_2) \in L^2 \\ S_{Act}(l_1) \neq S_{Act}(l_2) \quad OR \quad O_{Em}(l_1) \neq O_{Em}(l_2) \end{array} \quad (9)$$

- Distinguishability of evolutions:

$$\begin{array}{l} \forall (e_1, e_2) \in Evol^2 \\ l_U(e_1) \neq l_U(e_2) \quad OR \quad l_D(e_1) \neq l_D(e_2) \end{array} \quad (10)$$

- The upstream and downstream locations of each evolution are different. All evolutions represent changes of locations; there is no self-loop:

$$\forall (e) \in Evol \quad l_U(e) \neq l_D(e) \quad (11)$$

- Determinism of evolution[4]:

$$\begin{array}{l} \forall (e_1, e_2) \in Evol^2 \\ \text{if } l_U(e_1) = l_U(e_2) \\ \text{then } E_{Evol(I_G)}(e_1) \cdot E_{Evol(I_G)}(e_2) = 0 \end{array} \quad (12)$$

- For each location, there is no combination of inputs values which satisfy both the stability condition of this location and an evolution condition from this location:

$$\begin{array}{l} \forall l \in L \\ E_{Stab(I_G)}(l) \cdot \left(\sum_{\substack{e\in Evol \\ l_U(e)=l}} E_{Evol(I_G)}(e) \right) = 0 \end{array} \quad (13)$$

- For each location and for each combination of inputs values, the behavior is completely defined (either the location is stable, or there exists an evolution from this location):

[4] In the Boolean equations, 0 means False and 1 True.

$$\forall l \in L$$
$$E_{Stab(I_G)}(l) + \left(\sum_{\substack{e \in Evol \\ l_U(e)=l}} E_{Evol(I_G)}(e) \right) = 1 \quad (14)$$

- There is no transient location:

$$\begin{array}{l} \forall (e_1, e_2) \in Evol^2 \\ \text{if } l_D(e_1) = l_U(e_2) \\ \text{then } E_{Evol(I_G)}(e_1) \cdot E_{Evol(I_G)}(e_2) = 0 \end{array} \quad (15)$$

Last, the definition of the stable location must be consistent with the Grafcet evolution rules and actions definitions, i.e.:

- When the stability condition is true, no transition that is enabled for the corresponding situation of g can be fired (16).

- An output controlled by a continuous action in g belongs to the set of emitted outputs iff this continuous action is associated to an active step of the corresponding situation of g and the combinations of inputs values that satisfy the stability condition of the stable location satisfy the action condition too (17).

$$\begin{array}{l} \forall l \in L \\ \left(\prod_{s \in S_{Act}(l)} X(s) \right) \cdot \left(\prod_{\substack{s \in S_G(g) \\ s \notin S_{Act}(l)}} \overline{X(s)} \right) \cdot E_{Stab(I_G)}(l) \\ \cdot \left(\sum_{\substack{t \in T_G(g) \\ S_U(t) \subset S_{Act}(l)}} E_{Cond(I_G, S_G)}(t) \right) = 0 \end{array} \quad (16)$$

$$\begin{array}{l} \forall l \in L \\ \left(\prod_{s \in S_{Act}(l)} X(s) \right) \cdot \left(\prod_{\substack{s \in S_G(g) \\ s \notin S_{Act}(l)}} \overline{X(s)} \right) \cdot E_{Stab(I_G)}(l) \\ \cdot \left(\sum_{\substack{o \in O_{Em}(l) \\ o \in O_C}} \overline{E_{Emit(I_G, S_G)}(o)} \right. \\ \left. + \sum_{\substack{o \notin O_{Em}(l) \\ o \in O_G}} E_{Emit(I_G, S_G)}(o) \right) = 0 \end{array} \quad (17)$$

4.3. Construction of the SLA of a given Grafcet model

As previously pinpointed, all evolutions of an SLA come from a change of inputs values. However, two kinds of evolutions may occur in an SLA:

- Evolutions that correspond to the firing of a sequence of sets of simultaneously fired transitions in the Grafcet.

- Evolutions that do not correspond to the firing of Grafcet transitions but only to the change of the emitted outputs.

In the first case, the set of active steps $S_{Act}(l)$ is changed; this is not true in the second one where only O_{Em} and $E_{Stab(I_G)}$ are modified. The SLA of a given Grafcet model g is then built from the initial stable location by determining all these evolutions.

From a stable location $l = (S_{Act}, O_{Em}, E_{Stab(I_G)})$, computation of evolutions conditions is performed by symbolic calculus on Boolean expressions on inputs and steps activity variables; if two sets S_1 and S_2 of inputs values combinations are defined by expressions Exp_1 and Exp_2, then sets $S_1 \cap S_2$, $S_1 \cup S_2$, and $S_1 \setminus S_2$ are respectively represented by $(Exp_1 \cdot Exp_2)$, $(Exp_1 + Exp_2)$ and $(Exp_1 \cdot \overline{Exp_2})$. Locations stability and outputs emission conditions will be computed in a similar manner. This solution avoids combinatorial explosion and is well suited to Grafcet models where transitions and actions conditions are Boolean expressions.

SLA evolutions due to the firing of a sequence of sets of simultaneously fired Grafcet transitions

These evolutions are determined from a stable location $l = (S_{Act}, O_{Em}, E_{Stab(I_G)})$, by:

1. Looking for the set of fireable Grafcet transitions and all subsets of simultaneously fireable transitions, from the situation S_{Act}.

2. Determining the situations that are reached when all subsets of simultaneously fireable transitions are fired; if the new situation is transient, then 1 and 2 are to be reiterated until a stable situation is found[5].

3. Finding the set of emitted outputs for the stable situation reached by the sequence of firings.

These three computations are detailed as follows.

1. A Grafcet transition is fireable when it is enabled and its associated condition true. Hence, the set of enabled situations t from situation $S_{Act}(l)$ of a location l is first computed:

$$T_{Enab}(S_{Act}(l)) = \{t \in T(g) \quad | \quad S_U(t) \subset S_{Act}(l)\} \quad (18)$$

[5]When the sequence of firings is infinite, e.g. a cycle of evolutions from transient situation to transient situation, the Grafcet is said 'not sound' and is to be redesigned; no SLA is generated.

The set of fireable transitions from situation $S_{Act}(l)$ is the subset of $T_{Enab}(S_{Act}(l))$ for which inputs and step activity variables values satisfy the transition condition of a transition t. This set $T_{Fire}(S_{Act}(l))$ is defined as follows:

$$T_{Fire}(S_{Act}(l)) = \left\{ t \in T_{Enab}(S_{Act}(l)) \quad | \quad \left(\prod_{s \in S_{Act}(l)} X(s) \right) \cdot \left(\prod_{\substack{s \in S_G \\ s \notin S_{Act}(l)}} \overline{X(s)} \right) \cdot (E_{Cond(I_G,S_G)}(t)) \neq 0 \right\} \tag{19}$$

For simplicity reasons, in what follows:

- $E^{S_{Act}(l)}_{Fire(I_G)}(t)$ denotes the Boolean expression on inputs that represents the combinations of inputs for which transition t can be fired from situation $S_{Act}(l)$. This Boolean expression is easily obtained from the transition condition of t by substituting all steps activity variables by their corresponding value (True/False) according to $S_{Act}(l)$.

$$E^{S_{Act}(l)}_{Fire(I_G)}(t) = \sum_{\substack{c \in C_G \\ S_{Act}(l) = S_U(t)}} E_{Cond(I_G,S_G)}(t) \quad \text{where} \begin{cases} \forall s \in S_{Act}(l) & X(s) = True \\ \forall s \notin S_{Act}(l) & X(s) = False \end{cases} \tag{20}$$

- The term I_G is suppressed in all expressions denoted $E_{...}$ because they only depend on inputs.

With these notations, the set of fireable Grafcet transitions from situation $S_{Act}(l)$, $T_{Fire}(S_{Act}(l))$ is defined by:

$$T_{Fire}(S_{Act}(l)) = \left\{ t \in T_{Enab}(S_{Act}(l)) \quad | \quad E^{S_{Act}(l)}_{Fire}(t) \neq 0 \right\} \tag{21}$$

$SSFT(S_{Act}(l))$ denotes the set of all subsets SFT of simultaneously fireable transitions from $S_{Act}(l)$. This set is defined by:

$$SSFT(S_{Act}(l)) = \{ SFT \subset T_{Fire}(S_{Act}(l)) \quad | \quad E_{Evol}(SFT) \neq 0 \}$$

where:

$$E_{Evol}(SFT) = \left(\prod_{t \in SFT} E^{S_{Act}(l)}_{Fire}(t) \right) \cdot \left(\prod_{\substack{t \in T_{Fire}(l) \\ t \notin SFT}} \overline{E^{S_{Act}(l)}_{Fire}(t)} \right) \tag{22}$$

2. The set of situations reached when these subsets are fired from $S_{Act}(l)$ is defined by:

$$S^{S_{Act}(l)}_{SFT} = \left\{ \left(S_{Act}(l) \setminus \bigcup_{t \in SFT} S_U(t) \right) \cup \bigcup_{t \in SFT} S_D(t) \right\} \tag{23}$$

Among the set of situations $S^{S_{Act}(l)}_{SFT}$, the stable ones are situations *sit* such that expression (24) is satisfied (a given situation *sit* is reached from $S_{Act}(l)$ by firing a subset SFT and there is no enabled transition that is fireable from *sit*).

$$E_{Stab} = E_{Evol}(SFT) \cdot \overline{\sum_{t \in T_{Fire}(sit)} E^{sit}_{Fire}(t)} \tag{24}$$

The other elements of $S^{S_{Act}(l)}_{SFT}$ are transient situations which satisfy (25) (a given situation *sit* is reached from $S_{Act}(l)$ by firing an element of $SSFT$ and there is at least one enabled transition that is fireable from *sit*). Computation of the next sets of simultaneously fireable Grafcet transitions and reachable situations from *sit* is to be reiterated.

$$T^{E_{Evol}}_{Fire}(sit) = \left\{ t \in T_{Enab}(sit) | E^{sit}_{Fire}(t) \cdot E_{Evol} \neq 0 \right\} \tag{25}$$

3. For each stable situation, it is necessary to determine the emitted outputs. A continuous output o_C is emitted for a stable situation (sit, E_{Stab}) if and only if the emission condition of this output $E_{Emit}(o)$ satisfies:

$$E_{Emit}(o) \cdot \prod_{s \in sit} X(s) \cdot \prod_{\substack{s \in S_G \\ s \notin sit}} \overline{X(s)} \cdot E_{Stab} \neq 0 \tag{26}$$

As it was made for the firing conditions of transitions, $E^{(sit,E_{Stab})}_{Emit}(o)$ represents the combinations of inputs for which output o is emitted in the situation *sit*. This Boolean expression is obtained from $E_{Emit}(o)$ by substituting all the step activity variables by the corresponding value according to *sit*. The set $SSEO$ of all simultaneously emitted continuous outputs can be defined as follows:

$$SSEO(sit, E_{Stab}) = \{ SEO \subset O_C \quad | \quad E^{(sit,E_{Stab})}_{Emit}(SEO) \neq 0 \}$$

where $E^{(sit,E_{Stab})}_{Emit}(SEO) =$

$$\left(\prod_{o \in SEO} E^{sit}_{Emit}(o) \right) \cdot \left(\prod_{\substack{o \in O_C \\ o \notin SEO}} \overline{E^{sit}_{Emit}(o)} \right) \cdot E_{Stab} \tag{27}$$

The set of emitted outputs that are controlled by stored actions is determined by analyzing the sequence of firings between two successive stable situations. When a stored action is associated to a step that belongs to a (stable or transient) situation crossed by this sequence, the corresponding output is set or reset, according to the action type; when several stored actions on the same output are sequentially executed during a sequence, only the consequence of the latter one remains.

SLA evolutions without firing transitions

These evolutions correspond to changes of the set of emitted outputs while staying in the same situation. They are determined by finding the sets of simultaneously emitted continuous outputs obtained from the combinations of inputs that satisfy (28) (no transition enabled for $S_{Act}(l)$ can be fired).

$$E_{Stab} = \prod_{t \in T_{Fire}(S_{Act}(l))} \overline{E_{Fire}^{S_{Act}(l)}(t)} \qquad (28)$$

4.4. Illustration on the example

A software tool, named Teloco (TEst of LOgic COntrollers), has been developed during this work and is available at http://www.lurpa.ens-cachan.fr/isa/teloco/. This tool constructs automatically the SLA of a given Grafcet by using the above-presented definitions; it is also able to translate an SLA into an equivalent Mealy machine and to generate a test sequence from this machine, as it will be explained in the following two sections. This tool has been used for the example presented on figure 3; the SLA of this Grafcet contains 64 locations and 389 evolutions. The construction of this SLA lasts approximately 800 ms, what is quite reasonable.

Within this model, location ({F1,22,32},{ILG,VC,BM},[$\overline{TD} \cdot on \cdot \overline{z}$]) is reachable from the initial location; 13 evolutions are then possible from this particular location. Table 1 presents these evolutions, line by line. The first column gives the location that is reached when the evolution occurs, the second column contains the evolution condition (when the source location is active and this condition true, the evolution occurs), the third column the sequence of sets of simultaneously fired transitions from the source to the target location and the last one the continuous actions that are executed in the location (the suffix gives the name of the active step associated with this action and the name of the emitted output). It can be noted that the last line corresponds to an evolution without transition firing; only the set of emitted outputs is changed (output 'VC' is no more present). The other lines describe evolutions with sequences of firings. For the tenth evolution, for instance, two sets of transitions are successively fired; from the source location, transitions t22 and t32 are first simultaneously fired, leading to a transient situation, then transition t5 is fired and leads to the stable location ({F1,E40},{ILG,MR},[$on \cdot \overline{v}$]).

5. Translation of the SLA into an equivalent Mealy machine

This section aims to define the translation rules of an SLA constructed from a Grafcet into an equivalent Mealy machine. It is important to remind that a Boolean condition is associated with each evolution of an SLA (Table 1). In contrast, a Mealy machine is an event-based model, i.e. a transition of the machine is fired if an input event occurs and not if a Boolean condition is true. Therefore, the scientific issue to solve can be stated as follows: *"how to translate a state machine whose evolution conditions are defined by Boolean expressions into an event-based state machine, without any semantics loss ?"*. The answer to this concern is given below.

A Mealy machine M: $(I_M, O_M, S_M, s_{InitM}, \delta_M, \lambda_M)$ can be constructed from any stable location automaton SLA: $(I_{SLA}, O_{SLA}, L, l_{Init}, Evol)$. This Mealy machine behaves strictly as the SLA and is defined as follows:

- I_M: input alphabet. This alphabet contains $2^{|I_{SLA}|}$ elements. Each element i_i of this alphabet represents a different combination of logic variables of I_{SLA}. Each element i_i is associated with a distinct minterm[6] built on the variables of I_{SLA}. Let $m_I(i_i)$ be the minterm associated with the input event i_i. By construction, minterms are different and verify the following property:

$$\forall (i_i, i_j) \in I_M^2 \ \left[m_I(i_i) \cdot m_I(i_j) = 0 \right] \qquad (29)$$

For example, in the case of figure 3:
$m_I(i_{128}) = \overline{CS} \cdot TD \cdot \overline{a} \cdot \overline{b} \cdot \overline{dw} \cdot \overline{on} \cdot \overline{up} \cdot \overline{v} \cdot \overline{z}$
$m_I(i_{401}) = CS \cdot TD \cdot \overline{a} \cdot \overline{b} \cdot dw \cdot \overline{on} \cdot \overline{up} \cdot \overline{v} \cdot z$

- O_M: output alphabet. Similarly to I_M, this alphabet contains $2^{|O_{SLA}|}$ elements. Each element o_i of this alphabet represents a different combination of logic variables of O_{SLA}. Each element o_i is associated with a distinct minterm built on the variables of O_{SLA}. Let $m_O(o_i)$ be the minterm associated with the output event o_i.

[6] A minterm is an expression on Boolean variables that uses only the conjunction operator AND ($\cdot$) and the complement operator ($\bar{}$).

Reachable location	Evolution condition	Sequence of SFT	Continuous actions
({A3,22,32},{ILO,VC,BM},$[\overline{TD}\cdot\overline{z}]$)	$\overline{TD}\cdot\overline{on}\cdot\overline{z}$	$\langle$ {t2} $\rangle$	$\{a_{A3-ILO}, a_{22-VC}\}$
({A3,22,32},{ILO,BM},$[\overline{TD}\cdot(a+b)\cdot z]$)	$\overline{TD}\cdot(a+b)\cdot\overline{on}\cdot z$	$\langle$ {t2} $\rangle$	$\{a_{A3-ILO}\}$
({F1,S20,32},{ILG,BM},$[\overline{TD}\cdot on]$)	$\overline{TD}\cdot\overline{a}\cdot\overline{b}\cdot on\cdot z$	$\langle$ {t22} $\rangle$	$\{a_{F1-ILG}\}$
({F1,22,S30},{ILG,VC},$[on\cdot\overline{z}]$)	$TD\cdot on\cdot\overline{z}$	$\langle$ {t32} $\rangle$	$\{a_{F1-ILG}, a_{22-VC}\}$
({F1,22,S30},{ILG},$[(a+b)\cdot on\cdot z]$)	$TD\cdot(a+b)\cdot on\cdot z$	$\langle$ {t32} $\rangle$	$\{a_{F1-ILG}\}$
({A3,S20,32},{ILO,BM},$[\overline{TD}]$)	$\overline{TD}\cdot\overline{a}\cdot\overline{b}\cdot\overline{on}\cdot z$	$\langle$ {t2,t22} $\rangle$	$\{a_{A3-ILO}\}$
({A3,22,S30},{ILO,VC},$[\overline{z}]$)	$TD\cdot\overline{on}\cdot\overline{z}$	$\langle$ {t2,t32} $\rangle$	$\{a_{A3-ILO}, a_{22-VC}\}$
({A3,22,S30},{ILO},$[(a+b)\cdot z]$)	$TD\cdot(a+b)\cdot\overline{on}\cdot z$	$\langle$ {t2,t32} $\rangle$	$\{a_{A3-ILO}\}$
({F1,S20,S30},{ILG},$[on\cdot v]$)	$TD\cdot\overline{a}\cdot\overline{b}\cdot on\cdot v\cdot z$	$\langle$ {t22,t32} $\rangle$	$\{a_{F1-ILG}\}$
({F1,E40},{ILG,MR},$[on\cdot\overline{v}]$)	$TD\cdot\overline{a}\cdot\overline{b}\cdot on\cdot\overline{v}\cdot z$	$\langle$ {t22,t32},{t5} $\rangle$	$\{a_{F1-ILG}, a_{E40-MR}\}$
({A3,S20,S30},{ILO},$[v]$)	$TD\cdot\overline{a}\cdot\overline{b}\cdot\overline{on}\cdot v\cdot z$	$\langle$ {t2,t22,t32} $\rangle$	$\{a_{A3-ILO}\}$
({A3,E40},{ILO,MR},$[\overline{v}]$)	$TD\cdot\overline{a}\cdot\overline{b}\cdot\overline{on}\cdot\overline{v}\cdot z$	$\langle$ {t2,t22,t32},{t5} $\rangle$	$\{a_{A3-ILO}, a_{E40-MR}\}$
({F1,22,32},{ILG,BM},$[\overline{TD}\cdot(a+b)\cdot on\cdot z]$)	$\overline{TD}\cdot(a+b)\cdot on\cdot z$	$\varnothing$	$\{a_{F1-ILG}\}$

Table 1: Set of evolutions from location ({F1,22,32},{ILG,VC,BM},$[\overline{TD}\cdot on\cdot\overline{z}]$) for the Grafcet model presented figure 3

- S_M: set of states. A state of the Mealy machine M is associated with each location of the stable location automaton SLA. To make the translation easier to understand, the state of M and the associated location of SLA will be denoted in the same way. Thus, the set S_M is identical to the set L of SLA.

$$S_M \equiv L \tag{30}$$

- s_{InitM}: initial state. This state is associated with the initial location of SLA.

$$s_{InitM} \equiv l_{Init} \tag{31}$$

- The transition function δ_M and output function λ_M are defined on the basis sets I_{SLA}, O_{SLA}, L, $Evol$ and I_M, O_M, S_M. They must be defined so that the obtained Mealy machine is deterministic and completely specified, i.e.: [7]

$$\forall(s,i)\in S_M\times I_M \quad \left[\begin{cases} \exists!\delta_M(s,i)\in S_M \\ \exists!\lambda_M(s,i)\in O_M \end{cases}\right] \tag{32}$$

The transition function δ_M of the machine M is defined as follows:

$$\forall s\in S_M,\forall i\in I_M \left[\begin{array}{l} \text{if } \exists e\in Evol\left[\begin{cases} l_U(e)=s \\ E_{Evol(I_G)}(e)\cdot m_I(i)=m_I(i) \end{cases}\right] \\ \quad [\delta_M(s,i)=l_D(e)] \\ \text{else} \\ \quad [\delta_M(s,i)=s] \end{array}\right] \tag{33}$$

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

The transition function δ_M is completely specified since a value is defined for each 2-tuple $(s,i)\in S_M\times I_M$. This value is unique because the evolutions which follow a location l are mutually exclusive (see equation 12).

Construction of the output function λ_M relies on the property that outputs values only depend on the active location of the SLA. Then, the output function λ_M is defined as follows:

$$\begin{array}{l} \forall s\in S_M,\forall i\in I_M \quad [\lambda_M(s,i)=o_j] \\ \text{where } o_j\in O_M \text{ such that:} \\ \begin{cases} \forall v\in O_{Em}(\delta_M(s,i))\left[m_O(o_j)\cdot v=m_O(o_j)\right] \\ \forall v\in\{O_{SLA}-O_{Em}(\delta_M(s,i))\}\left[m_O(o_j)\cdot\bar{v}=m_O(o_j)\right] \end{cases} \end{array} \tag{34}$$

The proposed definitions of δ_M and λ_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 property holds for each situation of the SLA by construction.

According to the definitions of functions $\delta_M(s,i)$ and $\lambda_M(s,i)$, each 2-tuple $(s,i)\in S_M\times I_M$ verifies:

$$\begin{array}{l} \forall((s,i),(s',i'))\in(S_M\times I_M)^2 \\ [\delta_M(s,i)=\delta_M(s',i')\Rightarrow\lambda_M(s,i)=\lambda_M(s',i')] \end{array} \tag{35}$$

This result ensures minimality of the obtained Mealy machine. The translation rules that are detailed above have been implemented during this research in a software tool based on algorithm 1.

Last, the size of the machine is easily computable. The constructed Mealy machine contains indeed as

many states as there are locations in the SLA. The number of transitions only depends on the number of locations and input variables of the SLA (see relations (36)). It must be noted that the number of evolutions of the SLA has no influence on the size of the obtained Mealy machine.

$$\begin{cases} |states| = |L| \\ |transitions| = |L| \cdot 2^{|I_G|} \end{cases} \quad (36)$$

Thus, the Mealy machine that describes the behavior of the Grafcet depicted on figure 3 contains 64 states and 32,768 transitions, because the corresponding SLA includes 64 locations and owns 9 inputs. The translation of this SLA into its equivalent Mealy machine lasts approximately 300 ms, by using the tool Teloco. Figure 6 presents a part of the SLA and a part of the Mealy machine obtained from this example.

Algorithm 1 The Mealy machine construction algorithm

INPUT DATA: SLA: (I_{SLA},O_{SLA},L,l_{Init},$Evol$)
OUTPUT DATA: M: (I_M,O_M,S_M,s_{InitM},δ_M,λ_M)
for all $j = 0$ to $2^{|I_{SLA}|}$ **do**
  Build minterm $m_I(i_j)$ and add i_j to I_M
end for
for all $k = 0$ to $2^{|O_{SLA}|}$ **do**
  Build minterm $m_O(o_k)$ and add o_k to O_M
end for
for all $l \in L$ **do**
  if $l = l_{Init}$ **then**
    Associate s_{InitM} to l and add s_{InitM} to S_M
  else
    Associate s to l and add s to S_M
  end if
end for
for all $s \in S_M$ **do**
  for all $j \in I_M$ **do**
    Set the default value of $\delta_M(s, j)$: $\delta_M(s, j) := s$
    for all $e \in Evol$ **do**
      if $s = l_U(e)$ **then**
        if $E_{Evol(I_G)}(e) \cdot m_I(i_j) = m_I(i_j)$ **then**
          $\delta_M(s, j) := l_D(e)$
          break
        end if
      end if
    end for
    $\lambda_M(s, j) := o_k$ where $m_O(o_k)$ satisfies and only satisfies the emitted outputs condition for location l associated to s
  end for
end for

6. Test sequence generation

The Mealy machine which is constructed from an SLA represents all evolutions of the initial Grafcet. It is then possible to build, from this machine, a test sequence which can be used for conformance test of a logic controller which is supposed to implement the Grafcet specification. Several solutions are possible (Lee and Yannakakis (1996)). The method to obtain the minimum-length, while exhaustive, test sequence is called transition tour method (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 (Mei-Ko (1962), Edmonds and Johnson (1973) and Thimbleby (2003) for instance). The general formulation of this problem is the following: *"Find a minimum length closed path that visits each edge in the graph at least once"*. As the graph which describes the structure (states and transitions between states) of a Mealy machine is directed, but not weighted, the optimization problem is simplified.

The test sequence of the Mealy machine of the example (64 nodes and 32,768 arcs) contains 73,528 steps. This test sequence is obtained in approximately 1 s with the tool Teloco. If a step of this sequence is done every 20 ms during conformance test, the whole test execution will last approximately 25 minutes, what is reasonable.

7. Conclusions and prospects

This paper has shown that it is possible to construct automatically a test sequence for exhaustive conformance test of logic controllers from a non-timed Grafcet specification; this result can contribute undoubtedly to improve dependability of industrial logic controllers. The main contributions of this work are:

- the formal definition of Grafcet and the state model (SLA) that represents all stable states and evolutions of a Grafcet;
- a seamless test case generation method from Grafcet specification to test sequence. This method can be integrated in an industrial context since the computation time is not too long.

On-going works focus on improvement of the test sequence generation. As explained in section 6, the current approach is based on the transition tour method and provides a minimum-length, while exhaustive, sequence. This sequence is, in most cases, a MIC (Multiple Input Change) sequence, i.e. the values of several logic inputs may change from one test step to the

Reachable location	Evolution condition	Sequence of SFT	Continuous actions
({A1,1},{ILR},$[\overline{on}]$)	$\overline{dw} \cdot \overline{on} \cdot up$	⟨{t42},{t6},{t3}⟩	$\{a_{A1-ILR}\}$
({F1,1},{},$[(\overline{CS} + a + b + \overline{z} + dw + \overline{up}) \cdot on]$)	$(\overline{CS} + a + b + \overline{z}) \cdot \overline{dw} \cdot up \cdot on$	⟨{t42},{t6},{t3},{t1}⟩	$\varnothing$
({F1,E20,E30},{ILG,VA,BM},$[\overline{TD} \cdot \overline{a} \cdot on]$)	$CS \cdot \overline{TD} \cdot \overline{a} \cdot \overline{b} \cdot \overline{dw} \cdot on \cdot up \cdot z$	⟨{t42},{t6},{t3},{t1},{t4}⟩	$\{a_{F1-ILG}, a_{E20-VC}\}$
({F1,E20,31},{ILG,VA,BM},$[TD \cdot \overline{a} \cdot on]$)	$CS \cdot TD \cdot \overline{a} \cdot \overline{b} \cdot \overline{dw} \cdot on \cdot up \cdot z$	⟨{t42},{t6},{t3},{t1},{t4},{t30}⟩	$\{a_{F1-ILG}, a_{E20-VC}\}$

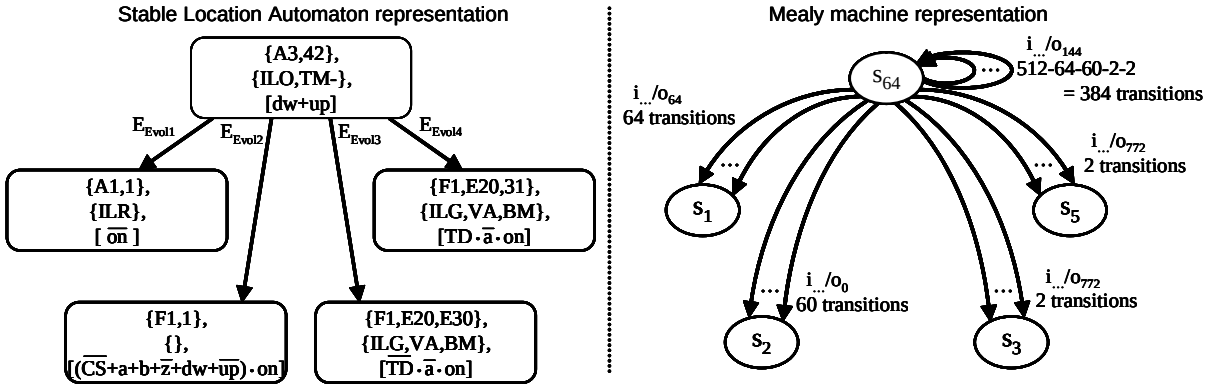


Figure 6: Representation of evolutions from location ({A3,42},{ILR},[$dw + up$]) using the SLA and the Mealy machine representations for the example given figure 3

other one. However, experiments have shown that it is better to use SIC (Single Input Change) sequences during test execution, to avoid synchronous inputs changes generated by the test-bench be seen as asynchronous by the implementation under test. Then, construction of a SIC, while exhaustive, test sequence deserves to be investigated. This sequence can be built from any specification that verifies the SIC-testability property defined in (Provost et al. (2010)). When this property does not hold, it matters to build a MIC test sequence which includes as few MIC test steps as possible; this is the aim of the current research.

References

Bierel, E., Douchin, O., Lhoste, P., 1997. Grafcet: from theory to implementation. Journal Européen des Systèmes Automatisés 31 (3), 543–559.

Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (Eds.), 2005. Model-based testing of reactive systems, Advanced lectures. Vol. 3472 of Lecture Notes in Computer Science.

da Silva Simão, A., Petrenko, A., Yevtushenko, N., 2009. Generating reduced tests for FSMs with extra states. Lecture Notes in Computer Science 5826, 129–145.

David, R., 1996. Grafcet: a powerful tool for specification of logic controllers. IEEE Transaction on Control Systems Technology 3 (3), 253–268.

Edmonds, J., Johnson, E. L., 1973. Matching, Euler tours and the Chinese postman. Mathematical Programming 5, 88–124.

Guéguen, H., Bouteille, N., 2001. Extensions of Grafcet to structure behavioural specifications. Control Engineering Practice 9 (7), 743 – 756.

IEC 60848, 2002. GRAFCET specification language for sequential function charts, 2nd Edition. International Electrotechnical Commission.

IEC 61131-3, 2003. Programmable controllers - Part 3: Programming languages, 2nd Edition. International Electrotechnical Commission.

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

Lhoste, P., Panetto, H., Roesch, M., 1993. Grafcet: from syntax to semantics. Automatique Productique Informatique Industrielle 27 (1), 127–141.

Massink, M., Latella, D., Gnesi, S., 2006. On testing UML statecharts. Journal of Logic and Algebraic Programming 69 (1-2), 1–74.

Mei-Ko, K., 1962. Graphic programming using odd or even points. Chinese Mathematics 1, 273–277.

Naito, S., Tsunoyama, M., 1981. Fault detection for sequential machines by transitions tours. In: Proceedings of the IEEE Fault Tolerant Computer Symposium. pp. 238–243.

Provost, J., Roussel, J.-M., Faure, J.-M., August 2010. SIC-testability of sequential logic controllers. In: Proceedings of 10th International Workshop on Discrete Event Systems (WODES 2010). pp. 203–208.
URL `http://hal.archives-ouvertes.fr/hal-00512767`

Thimbleby, H., 2003. The directed Chinese postman problem. Software – Practice & Experience 33 (11), 1081–1096.

Tretmans, J., 2008. Model based testing with labelled transition systems. Lecture Notes in Computer Science 4949, 1–38.

von Bochmann, G., Jourdan, G.-V., 2009. Testing k-safe Petri nets. In: Núñez, M., Baker, P., Merayo, M. G. (Eds.), TestCom/FATES - Testing of Software and Communication Systems. Vol. 5826 of Lecture Notes in Computer Science. Springer, pp. 33–48.