Formal models for conformance test of programmable logic controllers

Anaïs Guignard, Jean-Marc Faure

Formal models for conformance test of programmable logic controllers

Anaïs Guignard, Jean-Marc Faure

LURPA, Ecole Normale Superieure de Cachan
61 Avenue du President Wilson
94230 Cachan, France
firstname.lastname@lurpa.ens-cachan.fr

ABSTRACT. *Conformance test is a formal method that is aiming at checking whether an implementation, a Programmable Logic Controller (PLC) in this study, behaves as described by its specification. This method requires naturally a formal model of the specification. This paper shows how such a model can be built from a non-formal model in the IEC 60848 standardized language and the knowledge of the execution mode of this model by the PLC. Application to a simple example highlights the significance of the consideration of the execution mode for conformance test.*

RÉSUMÉ. *Le test de conformité est une méthode formelle dont le but est de vérifier qu'une implantation, ici un automate programmable industriel, se comporte comme décrit dans sa spécification. Cette méthode nécessite naturellement un modèle formel de la spécification. Ce papier montre qu'il est possible de construire un tel modèle à partir d'une spécification rédigée selon le langage non formel décrit dans la norme IEC 60848 ainsi que de la connaissance du mode d'exécution choisi pour l'automate. Une application à un exemple simple met en valeur l'importance du choix de ce mode d'exécution pour le verdict du test de conformité.*

KEYWORDS: *conformance test, Grafcet, location automaton, stability, transient evolution.*

MOTS-CLÉS : *test de conformité, Grafcet, automate des localités, stabilité, évolution fugace.*

---

1. Introduction

Programmable Logic Controllers (PLCs) are increasingly used to implement control and monitoring functions in critical systems like power plants, railway transport, etc. Hence, it really matters before commissioning a PLC to check whether it behaves as specified. Conformance test (IEC 1012, 2004) is a good solution to meet this objective. This validation technique assumes that the implementation under test, a PLC in this paper, is a black box with inputs-outputs; during the execution of the test, an input sequence is sent to the PLC and the resulting output sequence, response of the PLC to the input sequence, is compared to the expected output sequence (Figure 1). The set composed of this input sequence and the expected output sequence is called test sequence and must be built from the specification.

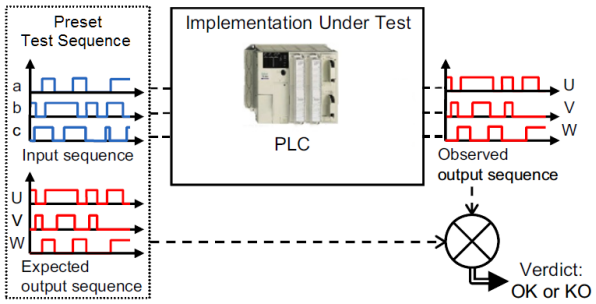

Figure 1. Conformance test principle

Construction of a test sequence when the specification is given in the form of a formal model with inputs and outputs, like a Mealy machine, has been addressed by many authors; a good synthesis of these works can be found in ((Lee, Yannakakis, 1996)). The specification of the behavior of PLCs is not represented by a formal model however, but by a non-formal model in a standardized language, like Grafcet ((IEC 60848, 2002)). This tailored-made language is very appreciated by numerous automation engineers because it is based on a graphical representation and permits to model parallelism, selection of sequences, synchronization, resource sharing. This paper focuses only on PLCs whose behavior is specified in this language while the results presented can be extended to other industrial specification languages, provided that their semantics can be completely defined.

To bridge the gap between the theoretical results on conformance test sequence generation and the industrial practices, ((Provost *et al.*, 2011b)) proposes a method to translate a Grafcet specification into an equivalent Mealy machine, without semantics loss. Nevertheless, this proposal supposes that the controller executes the Grafcet according to an algorithm based on stability search; in that case, the outputs are updated only if the Grafcet state is stable for the current values of its inputs, i.e. it cannot change without the value of at least one input is modified. However, most PLCs do not use this execution algorithm but interpret the Grafcet model without stability search; this execution mode guarantees indeed a shorter response time and a more frequent

output update what is really significant in particular for critical systems. Hence, the aim of this paper is to propose a more general method to build the formal model of a PLC that executes a Grafcet specification, whatever the execution mode.

Nevertheless, this proposal supposes that the controller executes the Grafcet according to an algorithm based on stability search; in that case, the outputs are updated only if the Grafcet state is stable for the current values of its inputs, i.e. it cannot change without the value of at least one input is changed. However, most PLCs do not use this execution algorithm but interpret the Grafcet model without stability search. This execution mode guarantees indeed a shorter response time and a more frequent output update what is really significant in particular for critical systems. Hence, the aim of this paper is to propose a more general method to build the formal model of a PLC that executes a Grafcet specification, whatever the execution mode.

The global approach that has been selected to generate a test sequence is depicted in Figure 2:

– The Grafcet model is first translated into a formal model, called *Location Automaton*, which represents the expected implementation of the specification. This translation requires the execution mode (with or without stability search) be defined as illustrated on this figure.

– Then, the Location Automaton is translated into an equivalent Mealy machine where all evolutions due to input changes are explicitly described.

– Finally, the test sequence can be built from this machine, by using for instance the transition-tour method ((Naito *et al.*, 1981)) to minimize the number of steps in the test sequence.

This paper considers mainly the first phase of this method; the other ones are widely detailed in ((Provost *et al.*, 2011b)). The construction of the Location Automaton for the two execution modes will be first explained; then, the test sequences built from these automata will be compared and discussed.

The semantics of the Grafcet specification language and the two execution modes of a Grafcet are reminded respectively in sections 2 and 3. Section 4 focuses on the construction of the formal model for the two execution modes of a PLC. These proposals are exemplified on a simple case from the literature in section 5. Section 6 is dedicated to the analysis of a part of the test sequence built from the two formal models and finally the concluding remarks and some perspectives are proposed in section 7.

2. Grafcet: Specification model

2.1. IEC 60848 Grafcet standard

This standard proposes a specification language, termed Grafcet, to describe the desired behavior of a logic system. A Grafcet model is a directed graph or a set of directed graphs with two kinds of nodes: steps and transitions. A step is represented

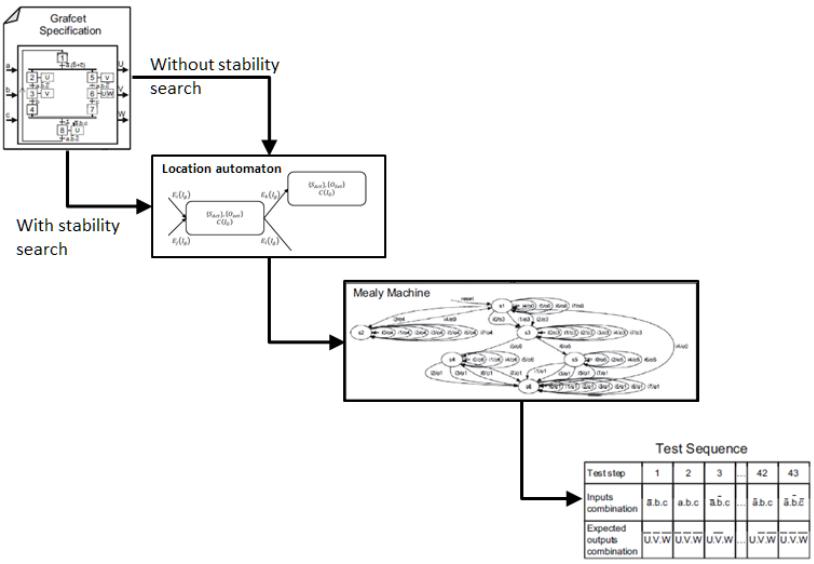

Figure 2. Test sequence generation method

by a square and defines a partial state of the system. A transition is represented by a horizontal line. A step can be linked only to transitions, and a transition can be linked only to steps. By default, the links are bottom-directed; if a top-directed link is to be introduced, an arrow must be added to this link to point out its direction.

A step can be active or inactive; then, a Boolean variable termed step activity variable is associated to each step. Moreover, one or several actions may be associated to each step; an action acts upon one internal or output variable. The set of the active steps of a Grafcet at a given time is called the *situation* of the Grafcet. Furthermore, a Boolean expression, called transition-condition, must be associated to each transition; this expression is defined on input variables, step activity variables or timed conditions.

Given this syntax, the evolutions of a model are described textually in the standard (IEC 60848, 2002) by a set of five rules:

– Rule n°1: *The initial situation, chosen by the designer, is the situation at the initial time.*

– Rule n°2: *A transition is said to be enabled when all immediately preceding steps linked to this transition are active. The firing of a transition occurs:*

- *when the transition is enabled,*
- *and when its associated transition-condition is true.*

When these two conditions are satisfied, the transition must be fired.

– Rule n°3: *The firing of a transition simultaneously provokes the activation of all the immediately succeeding steps and the deactivation of all the immediately preceding steps.*

– Rule n°4: *Several transitions which can be fired simultaneously are simultaneously fired.*

– Rule n°5: *If, during the evolution, an active step is simultaneously activated and deactivated, it remains active.*

It matters to underline that the standard focuses on the behavior of a specification model but not on the execution algorithm when this model is implemented in a PLC. Moreover, the syntax and semantics of the specification language are only described textually. (Lhoste *et al.*, 1993) and (Bierel *et al.*, 1997) have already pointed out this issue and proposed some worthwhile contributions, in the form of a static meta-model or algorithmic descriptions. Nevertheless these references do not propose a systematic way to construct, from a Grafcet model, a formal model of a PLC that executes a Grafcet.

The results presented in this paper will be illustrated on a simple case introduced in (David, Alla, 1992). The plant to control is a set of two tanks (Figure 3) that must be filled then emptied. The controller has five logic inputs (h_1, l_1, h_2, l_2, s) corresponding to position sensors and a push-button and four logic outputs (V_1, W_1, V_2, W_2) which order the opening (value True) and closing (value False) of valves; its expected behavior is described by the Grafcet of Figure 4.[1] From the initial step, the filling of the tanks starts when the button s is pushed. When a tank is completely filled (level h_1 or h_2 reached), it is emptied till the low level is reached. The initial step becomes active again only when the two tanks are empty (levels l_1 and l_2 reached).

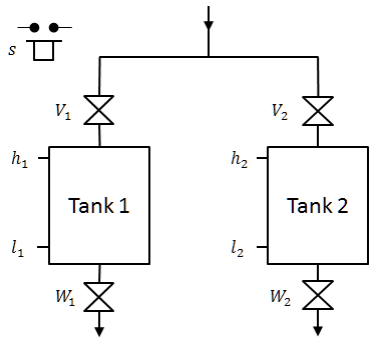

Figure 3. Tanks filling/emptying system

1. In this paper, $\overline{v}$ represents the negation of the Boolean variable v; $\wedge$ and $\vee$ represent respectively the conjunction and disjunction operators.

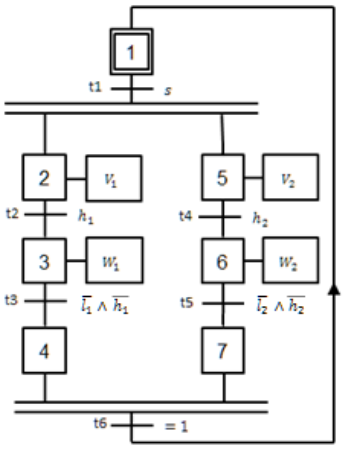

Figure 4. Specification of the controller

2.2. Formal model of the Grafcet language

Five hypotheses have been made to build this model:

– H1: The time-dependent elements of the Grafcet language (time dependent transition-condition, delayed and time-limited actions) are not considered.

– H2: There is neither rising edge nor falling edge of logic variables in the transition-conditions.

– H3: The specification model does not include any enclosing step, macro-step or forcing order.

– H4: The specification model does not include any source or pit step.

– H5: Input, output and internal variables are exclusively Boolean variables.

These hypotheses are aiming at facilitating the construction of the formal model and are reasonably realistic. The Grafcet elements forbidden by hypotheses H3 and H4 are found very rarely in industrial models and can be replaced by other Grafcet structures, even if forcing orders will generate partial order relations between graphs (Lesage, Roussel, 1993). It is also well-known that a transition whose condition includes an edge can be replaced by two transitions whose conditions include only input and step activity variables. The first hypothesis is the most restrictive but it is possible to represent a model with time-dependent elements by a model without time-dependent elements which is connected to timers. It matters to underline that these hypotheses hold for a model composed of only one connected graph, as this is the case in figure 4, or several connected graphs synchronized by step activity variables.

Under those hypotheses, a Grafcet G is formally defined by a $6-tuple$ $(I_G, O_G, S_G, T_G, A_G, S_{Init})$ where:

– I_G is the non-empty set of inputs of the logic system,

– O_G is the non-empty set of outputs of the logic system,

– S_G is the non-empty set of Grafcet steps, with X_G the set of associated step activity variables,[2]

– T_G is the non-empty set of Grafcet transitions,

– A_G is the set of Grafcet actions,

– S_{Init} is the set of the initial steps.

A transition $t \in T_G$ is defined by a $3-tuple$ $(S_U, S_D, F(I_G, X_G))$ where:

– S_U is the non-empty set of (immediately preceding) upstream steps,

– S_D is the non-empty set of (immediately succeeding) downstream steps,

– $F(I_G, X_G)$ is the transition-condition, Boolean expression on input and step activity variables.

The set of actions A_G is split in two sub-sets:

– the sub-set of continuous actions A_C. The output controlled by a continuous action is set only when a step to which this action is associated is active; when this step becomes inactive, the output is reset.

– the sub-set of stored actions A_S. The output controlled by a stored action is set and reset by this action when a step to which it is associated becomes active; when the step is deactivated, the output keeps its value (true/false if the action was set/reset).

Formalization of the evolution rules will be addressed in section 4; nevertheless, the formal definitions of two concepts: transient evolution and stable situation, which are depicted in the IEC 60848 standard ((IEC 60848, 2002), page 13) and are fundamental for this study, are to be given now.

DEFINITION 1. — *Transient evolution*
Let $S_{Act} \subset S_G$ be the situation (set of active steps) of a Grafcet G at a given moment. Let $T_{enab} \subset T_G$ be the set of enabled transitions in this situation S_{Act}.

$$T_{enab} = \{t \in T_G | t.S_U \subset S_{Act}\}^3 \quad (1)$$

2. The step activity variable of a step $s \in S_G$ will be noted X_s.
3. In this paper, an element b of a tuple A will be noted $A.b$.

Let $i_G \in I_G$ be a combination of values of input variables. An element $t_i \in T_{enab}$ is firable when:

$$t_i.F(i_G, X_G) = True \tag{2}$$

This transition must be fired and the new set of active steps becomes:

$$S_{New} = \{(S_{Act} \backslash t_i.S_U) \cup t_i.S_D\} \tag{3}$$

If there exists $t_j \in T_G$ such as:

$$t_j \notin T_{enab}, t_j.S_U \subset S_{New} \text{ and } t_j.F(i_G, X_G) = True \tag{4}$$

then, the sequence $t_i t_j$ is a transient evolution for the combination of values of input variables i_G and the set $\{t_i.S_D \cap t_j.S_U\}$ is the set of unstable steps.

This definition deals with a sequence of only two transition firings but may be extended to a sequence of more than two sequential firings or to sequences where several transitions are simultaneously fired.

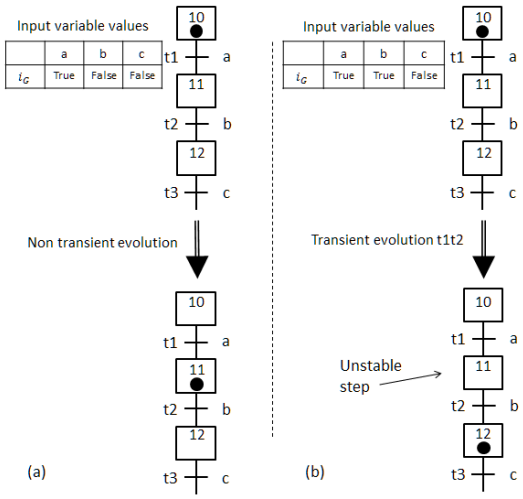

Figure 5. Non transient (a) and transient (b) evolutions

A simple example is proposed figure 5 where the combination of the initial values of input variables leads or not to a transient evolution of the Grafcet model.

Likewise, for a combination of values of input variables i_G, the situation of a Grafcet is said stable if no transition is firable from this situation for this combination of values.

DEFINITION 2. — *Stable situation*
Let $S_{Act} \subset S_G$ be the situation of a Grafcet G.
Let $T_{enab} \subset T_G$ be the set of enabled transitions in this situation:

$$T_{enab} = \{t \in T_G | t.S_U \subset S_{Act}\} \tag{5}$$

Let i_G be a combination of values of input variables. The Grafcet is said in a stable situation if:

$$\forall t \in T_{enab}, t.F(i_G, X_G) = False \tag{6}$$

Only a change of the value of at least one input variable (i_G becomes i'_G) may trigger the firing of one transition to leave this situation, because the values of the step activity variables are fixed if the situation is known.

3. Grafcet execution in a PLC

3.1. I/O scanning cycle

Most of programmable logic controllers (PLC) which are selected to implement a Grafcet specification are mono-task systems. The operation of a PLC is then based on a cyclic I/O scanning that can be periodic or not. A periodic I/O scanning cycle (figure 6) comprises 4 phases:

- Inputs reading (R),
- Internal and output variables computation (C),
- Outputs updating (U),
- Waiting time until the end of the period T.

The duration of the first and third phases is constant while that of the second phase may vary from one cycle to the other.

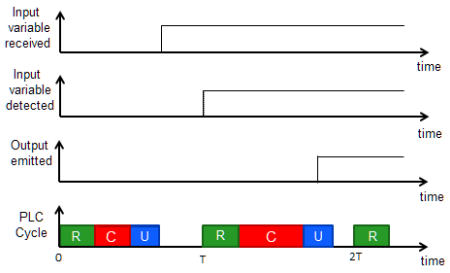

Figure 6. Periodic I/O scanning

The waiting phase is omitted when the cycle is aperiodic. In both cases, the cycle in controlled by a hardware or software timer, termed watchdog, which stops the operation and sets up the output values to ensure safety, when the duration of the cycle

exceeds a given threshold: T for a periodic cycle or a time defined by the designer for an aperiodic cycle. The main objective of this watchdog is to detect internal failures of the PLC and programming flaws so as to avoid the outputs be not updated during a too long time, what could lead to lose the correct control of the plant.

The computation phase is performed by a code in a standardized PLC programming language that is developed from the specification model and once one of the two execution modes described below has been selected.

3.2. Execution with or without stability search

The execution mode with stability search is described by Algorithm 1 in which a Boolean variable Stable is introduced to qualify the current situation of the Grafcet model. This variable is used in a stability search loop. Once a stable situation reached, the values of the outputs controlled by continuous actions are computed.

Algorithm 1 Grafcet execution with stability search

```
1: Stable := False;
2: while Stable = False do
3:     Determine the set of firable transitions;
4:     Determine the set of active steps;
5:     Compute the values of the outputs controlled by stored actions associated to
   the active steps;
6:     Determine the set of firable transitions;
7:     if the set of firable transitions is empty then
8:         Stable := True;
9:     end if
10: end while
11: Compute the values of the outputs controlled by continuous actions associated to
    the active steps;
```

The loop introduced in Algorithm 1 generates a variable duration of the variables computation phase of the I/O scanning cycle. This duration may even exceed the watchdog threshold for long transient evolutions; unexpected triggering of the watchdog is then possible when selecting the previous execution mode. This explains why the execution mode without stability search (Algorithm 2) is commonly selected for industrial applications.

Algorithm 2 Grafcet execution without stability search

```
1: Determine the set of firable transitions;
2: Determine the set of active steps;
3: Compute the values of the outputs controlled by stored actions;
4: Compute the values of the outputs controlled by continuous actions;
```

An illustration of this phenomenon of time consumption is presented through an evolution of the Grafcet given figure 1. The figure 7 shows that from a given situation and values of input variables (figure 7.a)), a simple change of one input value may lead to several successive firings of transitions then to several computation loops of the algorithm 1. Indeed, from this configuration, if the value of $l2$ turns from True to False, the transitions t_5, t_6 and t_1 will be successively fired to reach the stable situation shown in 7.b). This highlights that, even for a basic example, the computation phase with stability search, and consequently the PLC cycle, may last more than expected.

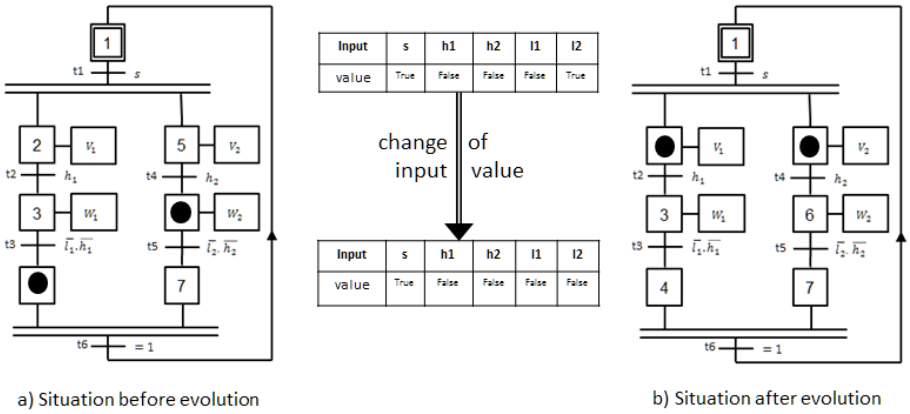

Figure 7. Transient evolution including three successive firing of transitions

4. Construction of formal models

The aim of this section is to propose an algorithm to construct a formal model that will be used for test sequence generation, from the Grafcet specification model and the knowledge of the execution mode. The outcome of the algorithm will be a location automaton (LA) (Provost *et al.*, 2011a), class of finite state automaton that permits to represent all Grafcet concepts; moreover, a location automaton can be easily translated, without semantics loss, in a Mealy machine from which a test sequence can be built.

In an intuitive manner, a state of this automaton, called location, gathers the concepts of current situation of the Grafcet, set of outputs that are set in this situation and stability condition. Every transition between two locations, called evolution, is labeled with a Boolean expression on the input variables, named firing condition; when this expression is true and the location that is the source of this evolution active, the evolution must be fired. The stability condition of a location is the negation of the disjunction of the firing conditions of all evolutions which start from this location.

4.1. Definitions

A location automaton LA is defined as a $5-tuple$
$LA = (I_{LA}, O_{LA}, L, l_{Init}, Evol)$ where:

- $I_{LA} = I_G$ is the set of logic inputs,
- $O_{LA} = O_G$ is the set of logic outputs,
- L is the set of locations,
- l_{Init} is the initial location,[4]
- $Evol$ is the set of evolutions.

Each location $l \in L$ is defined by a $3-tuple$ $(S_{Act}, O_{Set}, C(I_G))$ where:

- S_{Act} is the current situation of Grafcet G,
- O_{Set} is the subset of outputs of G which are set in this situation (the other outputs are reset),
- $C(I_G)$ is the stability condition of the location, Boolean expression on I_G; when this expression is true, no transition of G is firable.

Each evolution $evol \in Evol$ is defined by a $3-tuple$ $(l_U, l_D, E(I_G))$ where:

- $l_U \in L$ is the upstream location of the evolution,
- $l_D \in L$ is the downstream location of the evolution,
- $E(I_G)$ is the firing condition of the evolution, Boolean expression on the input variables; when the upstream location is active and this expression true, the evolution must be fired.

The graphical representation of the elements of a location automaton is given Figure 8.

As a LA is deterministic, the firing conditions of the evolutions that start from a location are exclusive:
$\forall l \in L, \forall evol_i \in Evol, \forall evol_j \in Evol, i \neq j$ where:
$evol_i \in \{Evol | evol_i.l_U = l\}$ and
$evol_j \in \{Evol | evol_j.l_U = l\}$ as

$$(evol_i.E(I_G)) \wedge (evol_j.E(I_G)) = False \tag{7}$$

4. There is only one initial location because a LA represents the expected behavior of a controller then must be deterministic.

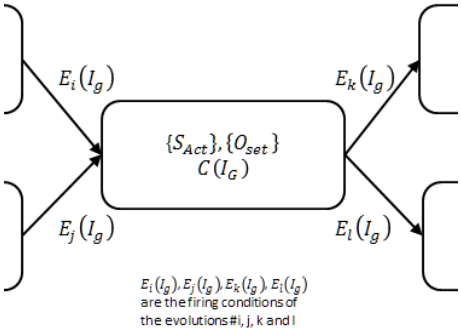

Figure 8. Representation of a location and evolutions

And the stability condition of a location $l \in L$ is:

$$l.C(I_G) = \overline{\sum_{evol_k.l_U = l} evol_k.E(I_G)} \tag{8}$$

4.2. Formalization of the evolution rules of the Grafcet language

The evolution rules of the location automaton built from a Grafcet model must be consistent with those given in section 2.1. Then it matters to express formally these textual explanations before constructing the equivalent location automaton.

The Rule n°2 indicates the conditions to fire a Grafcet transition: all the immediately upstream steps must be active, i.e. the transition is enabled, and the associated transition-condition must be True. Then, the firing condition of a transition for a given situation can be stated formally from the transition-condition by replacing the step activity variables by their values for this situation:
Let S_{Act} be a Grafcet situation,
the firing condition of a transition $t \in T_G$ is:

$$t.F(I_G) = t.F(I_G, X_G) \tag{9}$$

$$with \begin{cases} \forall s \in S_{Act} & X_s = True \\ \forall s \notin S_{Act} & X_s = False \end{cases}$$

Therefore, a transition t is firable for a given combination of values of the input variables if it is enabled and its firing condition is True for this combination, i.e.:
$t \in T_{enab}$ and $\exists i_G \in I_G$ such as:

$$t.F(i_G) = True \tag{10}$$

The Rule n°4 specifies that several transitions may be fired simultaneously. The firing condition of a set of transitions $F_\tau(I_G)$ must then be defined as follows.

Let $\tau \subseteq T_{enab}$ be a subset of the set of enabled transitions. The firing condition of this set of transitions is the conjunction of the firing conditions of each transition of this subset and the negation of the firing conditions of the enabled transitions that do not belong to this subset.

$$F_\tau(I_G) = \prod_{\substack{t \in T_{enab} \\ t \in \tau}} t.F(I_G) \wedge \prod_{\substack{t \in T_{enab} \\ t \notin \tau}} \overline{t.F(I_G)} \tag{11}$$

The relation (11) is not true whatever τ, however, because it is not always possible to find a combination i_G that satisfies this relation. A subset of simultaneously firable transitions SFT, for a given situation S_{Act}, is a subset that satisfies (11) for at least one combination of values of the input variables.

$$\exists i_G \in I_G, F_{SFT}(i_G) = True \tag{12}$$

This set may be not unique; the set of the subsets of simultaneous firable transitions, noted $SSFT$, is to be introduced:

$$SSFT = \{SFT \subseteq T_{enab} \mid \exists i_G \in I_G, F_{SFT}(i_G) = True\} \tag{13}$$

When a set of transitions SFT is fired, all upstream steps are deactivated and all downstream steps are activated according to the Rule n°3; then, the step activity variables are updated in the formal model. Nevertheless, concurrency between activation and deactivation may occur for some steps. This is solved by the Rule n°5 that means that activation has priority over deactivation. To respect this rule, the update of the step activity variables must merely treat the set operations after the reset ones.

4.3. Definitions of the locations and evolutions of the LA from the Grafcet

As the sets of inputs and outputs of the Grafcet and the equivalent location automation are identical by definition, only the sets of locations and evolutions are to be created.

A location is defined by a situation, a set of outputs that must be set in this situation and a stability condition. The set of locations L is then defined from the set of situations S_{Act} by considering every situation of the Grafcet model, if the selected execution mode is without stability search, and only the stable situations if it is assumed that the PLC executes the specification model without stability search. The set of outputs to set is derived from the set of the Grafcet actions associated to the steps that define the situation. Last, the stability condition $C(IG)$ is obtained from the firing conditions of the evolutions that start from the location as stated by (8); it matters then to formalize these conditions.

An evolution of a location automaton describes the firing of a set of transitions SFT, if no stability search is considered, or a sequence of $SFTs$ otherwise. Thus,

the set of evolutions $Evol$ is defined from the sets $SSFT$ for every situation of the corresponding Grafcet. The upstream location of an evolution is the location associated to the current situation and its downstream location is the one associated to the new situation that is reached once the evolution fired.

When every situation is considered (no stability search), the evolution condition is merely the firing condition of the SFT that is represented by the evolution:

$$E(I_G) = F_{SFT}(I_G) \tag{14}$$

When only stable situations are kept (stability search), the evolution condition is the conjunction of the firing conditions of the sequence of $SFTs$ that is represented by the evolution:

$$E(I_G) = \prod_{SFT \in \sigma_{SFT}} F_{SFT}(I_G) \tag{15}$$

Where σ_{SFT} is a sequence of $SFTs$ between two stable situations.

4.4. Construction of the location automaton

4.4.1. Execution without stability search

This construction may be described in a synthetic manner by the Algorithm 3 that defines the sets of inputs and outputs of the location automaton, its initial location and the outputs that are set in this location. It also calls the recursive function $LocationCondition(l)$ detailed by the Algorithm 4.

The aim of the function $LocationCondition(l)$ is to compute the stability condition of a location l. This condition is obtained by:

– Determining the sets of $SFTs$ for the current situation (function $CreateSFT(S_G)$),

– Updating the values of the step activity variables, i.e. defining the new situation, when a set SFT is fired (function $StepUpdate(S_G, SFT)$),

– Updating the values of the output variables in this new situation (function $EmitOut(S_G)$),

– Creating a location from this new situation and output values (function $CreateLocation(S_G, O_G)$),

– Creating the evolution from the the old location to the new reached location, labeled with the expression $E(I_G)$ (funtion $CreateEvol(L, L, E(I_G))$).

The function is then recalled till all locations and evolutions have been found; the stability conditions are computed from the firing conditions of the evolutions according (8).

Only the update of the step activity variables is detailed in this paper (Algorithm 5) for room reasons; it may be noted that this algorithm satisfies the Rule n°5 of

the Grafcet language (an active step that is simultaneously activated and deactivated remains active).

Algorithm 3 Construction of the equivalent location automaton

function BUILDLA(Grafcet)
  $I_{LA} = I_G$;
  $O_{LA} = O_G$;
  $Oset_{Init} = EmitOut(Sinit)$;
  $L_{init} = CreateLocation(S_{Init}, Oset_{Init})$;
  $L_{init}.C = LocationCondition(L_{init})$;
  return: Location Automaton
end function

Algorithm 4 Recursive function to obtain the stability condition of a location

function LOCATIONCONDITION(l) ▷ After having defined all the possible evolutions from a location, determines the stability condition of this location
  $ActSteps = l.S_{Act}$; ▷ Active Grafcet steps for the current location
  $NotC = False$ ▷ Negation of $C(I_g)$
  for $t \in T_G$ **do**
    Calculation of the firing condition of each enabled transition
  end for
  $SSFT = CreateSFT((ActSteps))$
  for $SFT \in SSFT$ **do** ▷ Determination of the new location reached for each evolution
    $NewS = StepUpdate(ActSteps, SFT)$
    $E = F_{SFT}$;
    $NewO = EmitOut(NewS)$;
    if The reached location already exists **then**
      $CreateEvol(l, l_{new}, E)$; ▷ with l_{new} the location reached
    else ▷ Defines the new location and possible evolutions
      $NewL = CreateLocation(NewS, NewO)$;
      $NewL.C = LocationCondition(NewL)$;
      $CreateEvol(L, NewL, E)$;
    end if
    $NotC = NotC + E$; ▷ Updates the stability condition of the location
  end for
  $C = \overline{NotC}$;
  return: C
end function

Algorithm 3 can be applied to a Grafcet model composed of only one connected graph or a set of several connected graphs which are synchronized by step activity variables but are not hierarchically ordered (flat model). However, some PLCs require that the set of connected graphs, when this structure is selected, be ordered, i.e. that an

Algorithm 5 Function which updates the set of active steps after the firing of a set of simultaneously firable transitions

```
1: function StepUpdate(S_Act, SFT)
2:     NewSteps = S_Act                          ▷ Initial situation
3:     for t ∈ SFT do
4:         NewSteps = NewSteps/t.S_U             ▷ Deactivation
5:     end for
6:     for t ∈ SFT do
7:         NewSteps = NewSteps ∪ t.S_D           ▷ Activation
8:     end for
9:
10:    return: NewSteps                          ▷ Final situation
11: end function
```

order relation be defined to execute sequentially the different graphs. In that case, the computations of the evolutions must integrate the hierarchy introduced by this order relation.

The upper bounds of the numbers of transitions and situations of the Grafcet model, as well as locations and evolutions of the LA can be easily given. If $s = |S_G|$ is the number of Grafcet steps and $n = |I_G|$ the number of input variables, these bounds are respectively:

– $|T_G| = 2^s * 2^n$ transitions; this bound is reached when as many transitions as there are combinations of the input values start from every situation of the Grafcet.

– $|L| = 2^s$; this bound is also that of the number of situations of the Grafcet, by definition.

– $|Evol| = 2^s * 2^n$; 2^n evolutions start from every location in that case.

Nonetheless, it must be underlined that these bounds are extremely pessimistic. Real Grafcet models are compact representations of control laws which have been designed by competent automation engineers; hence, the transition conditions of the transitions which start from any step correspond generally to sets of combinations of the input values and not only one, for instance. This remark will be illustrated in section 5.

Concerning the algorithm, the function LocationCondition (Algorithm 4) will be called for every location. In this function, a first loop (line 4.) on transitions is performed with performs an elementary operation. Then (line 8.), an other loop on simultaneous firable transitions, which corresponds to evolutions, starts. Inside this loop, on line 10., an operation containing a loop on all transitions is done. All the other operations are elementary ones.

The figure 9 details all the terms of the complexity. In this expression, a product of two terms means that the second is inside the loop described by the first and an addition of two terms means that they are following operations.

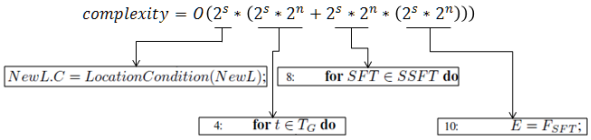

Figure 9. Origin of terms of the complexity equation

This expression can be reduced to:

$$\begin{aligned} complexity &= O(2^s * (2^s * 2^n + 2^s * 2^n * (2^s * 2^n))) \\ &= O(2^{2s+n} + 2^{3s+2n}) \\ &= O(2^{3s+2n}) \end{aligned} \tag{16}$$

4.4.2. Execution with stability search

The location automaton which corresponds to an implementation without stability search contains all the information required to construct the location automaton that models an implementation with stability search. This last automaton can be indeed built from the model obtained at the previous section by removing the unstable locations and merging the successive evolutions that represent a transient evolution. This is performed by Algorithm 6.

To detect whether an evolution leads to a stable or unstable location, this algorithm first checks that the evolution condition of the evolution is included in the stability condition of its downstream location; if this is not the case, this evolution is part of a transient evolution and will be merged with other consecutive evolutions of the initial automaton to give rise to a unique evolution of the location automation with stability search. The condition of this evolution is then constructed by iterating the computation until a stable location is reached.

5. Illustrative example

The algorithms presented in the previous section have been applied to the case study presented in section 2.1. The formal models that correspond to the two execution modes are depicted Figure 10 and Figure 11. These figures show clearly the main differences between the two models:

– The model without stability search contains one more location than the model with stability search. This location corresponds to the situation {4,7} in the initial Grafcet which is always unstable because the transition-condition of t_6 is always True; when both steps 4 and 7 are active, the firing condition of t_6 is True whatever the combination of the input values.

– The model with stability search contains (eleven) more evolutions. These evolutions correspond to transient evolutions (sequences of firings of SFTs); that kind of evolution is only possible with this execution mode.

Algorithm 6 Construction of the location automaton with stability search

1: **for** each location L **do**
2: $EvolList$ = evolutions from L
3: **while** $EvolList$ is not empty **do**
4: Pick $evol$ from $EvolList$
5: $dest = evol.l_D$
6: **if** $dest.C \wedge evol.F \neq dest.C$ **then** ▷ If there is a transient evolutions
7: $exp = evol.F \wedge \overline{dest.C}$
8: **for** each $evol_D$ as $evol_D.l_U = dest$ **do**
9: **if** $evol_D.F \wedge exp \neq False$ **then** ▷ Detection of the successively firable evolutions
10: $newEvol.l_U = L$ ▷ Creation of the new evolution
11: $newEvol.l_D = evol_D.l_D$
12: $newEvol.F = evol_D.F \wedge exp$
13: Add $newEvol$ to $EvolList$ ▷ Operation must be repeated for longer transient evolutions
14: **end if**
15: **end for**
16: $evol.F = evol.F \wedge \overline{exp}$ ▷ Suppression of the non stable part of the evolution
17: **end if**
18: **end while**
19: **end for**

These remarks can be generalized whatever the specification model. The number of locations of the automaton without stability search is always equal or greater than the number of locations of the automaton with stability search, because only the stable situations of the Grafcet are considered when building the last one. On the opposite, no definite conclusion can be drawn for the number of evolutions which depends on both the number of locations and the number of transient evolutions in the Grafcet.

These two examples illustrate the remark on the size of a location automaton which has been made at the end of 4.4.1. For the considered Grafcet (7 steps and 5 inputs), the upper bounds of the numbers of locations and evolutions of the equivalent LA are respectively $2^7 = 128$ and $128 * 2^5 = 4096$; these values are far greater than those obtained (11 and 19 for the first automaton and 10 and 30 for the second).

6. Location automaton for conformance test

A location automaton may be translated, without semantics loss, into an equivalent Mealy machine from which it is possible to construct a test sequence that covers at least once each transition of the machine as shown in (Provost *et al.*, 2011b); the test sequence thus depends on the initial location automaton. If this automaton models the expected behavior of a PLC, correct test verdicts will be obtained if and only if

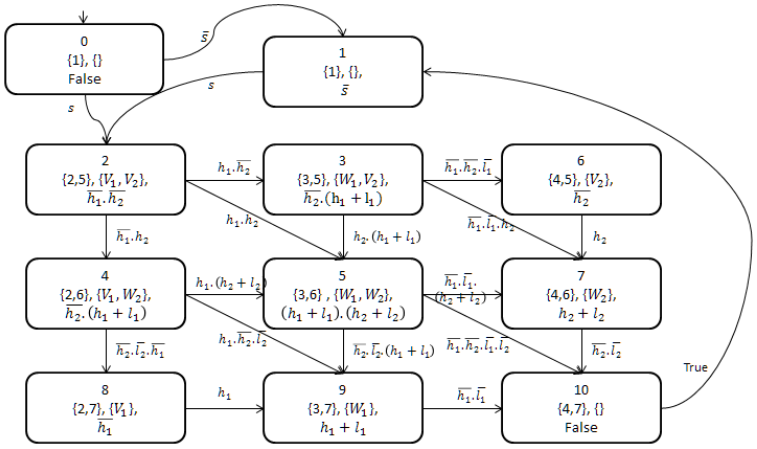

Figure 10. Formal model of a PLC that executes the Grafcet of figure 4 without stability search

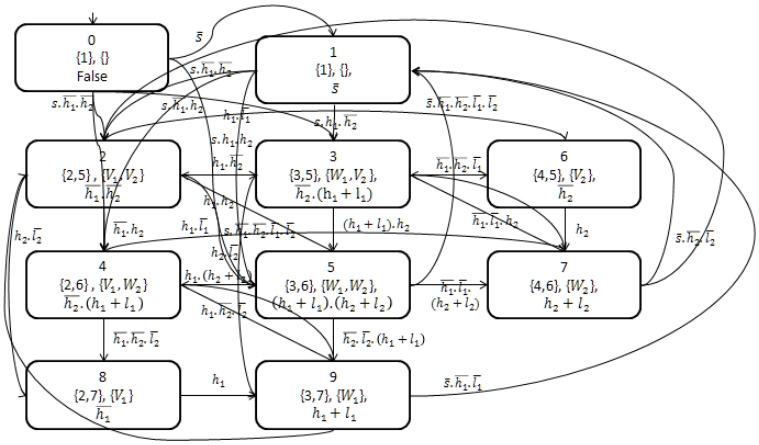

Figure 11. Formal model of a PLC that executes the Grafcet of figure 4 with stability search

the appropriate automaton, formalizing an execution mode without or with stability search, is selected, as shown below.

The table 1 presents the main differences between the models generated assuming that the execution is with or without stability search. Some observations may be done:

– The number of states of the Mealy machine is equal to the number of locations of the location automaton minus one. This slight size reduction is obtained by merging the initial state, derived from the initial location, with the state that corresponds to the location whose situation and emitted outputs are identical to those of the initial

location (1 in the example). This is possible because a state of this machine does not store indeed any information, like the stability condition or the emitted outputs. At initialization, the stability condition of the initial location: False in the example, which means that all input values are False, is not necessarily satisfied and if not, an evolution (towards the location 1 or 2 in the example) will occur. An equivalent behavior in the Mealy machine is obtained by merging the two states and removing the transitions between them.

– The number of transitions of the Mealy machine is equal to $t = m * 2^n$ where m is the number of states and n the number of input variables, From any state, there are as many transitions as there are combinations of input variables.

– The number of test steps in the test sequence built from the automaton without stability search is greater than that of the test sequence obtained from the automaton with stability search because a transient evolution of the Grafcet corresponds to several evolutions in the first model and only one in the second. The reader is reminded at this point that the aim of the Transition-Tour method is to build a test sequence that crosses at least once each transition of the Mealy machine while minimizing the number of test steps.

Table 1. Characteristic variables with or without stability search

	Location automaton		Mealy machine		Test sequence
	Locations	Evolutions	States	Transitions	Steps
With stability search	10	30	9	288	283
Without stability search	11	19	10	320	446

The table 2 presents a sequence of three test steps that has been built from the location automaton presented figure 11, which means that the controller is supposed to execute the Grafcet specification with stability search. The three test steps correspond respectively to an evolution from location 0 to location 1, then from location 1 to location 3, and finally from location 3 to location 5. In this table, 0 means that the variable is False and 1 means that the variable is True. The duration of each test step is greater than the maximal cycle time of the controller, a mandatory condition to insure that each change of the combination of the input values is detected by the PLC.

If this sequence is used to test a controller which executes the same specification model but without stability search (usual industrial case), and assuming that the controller executes correctly the automaton presented figure 10, the test verdicts will be the following ones for each test step:

– For the first test step, the active location becomes the location 1 and the observed combination of output values on the first step is O(1) = (0; 0; 0; 0) is the same as the expected one; the test is positive.

– For the second test step, the active location becomes the location 2 and the observed combination of output values O(2) = (1; 1; 0; 0) is different from the expected one; the test is negative, which was easily predictable because there is no evolution between locations 1 and 3 in the model of figure 10.

Table 2. Results of test sequence execution

		Test step		
		Step 1	Step 2	Step 3
Input sequence	s	0	1	0
	h_1	0	1	0
	h_2	0	0	1
	l_1	1	1	1
	l_2	1	1	1
Expected output sequence	V_1	0	0	0
	V_2	0	1	0
	W_1	0	1	1
	W_2	0	0	1
Observed output sequence	O(i)= V_1	0	1	1
	V_2	0	1	0
	W_1	0	0	0
	W_2	0	0	1
			or	
			1 0	0
			1 1	0
			0 1	1
			0 0	1

As the Grafcet is implemented without stability search, the transient evolution will be at least 2 cycles long because it is composed by two successive firing of transitions (t1 and t4). The second step corresponds to the firing of the first transition t1. On the next PLC cycle, either there is no input changes and the LA will reach its stable location, or the input values have been updated and taken in count for the next firing of transitions. Two cases are then possible according to the duration of a test step:

– If the duration of a test step is smaller than two PLC cycles, the location 4 is reached at the third step and the observed combination of output values is O(3) = (1; 0; 0; 1).

– Else, the automaton evolves from location 2 to location 3 without any change of the combination of input values; the observed combination of output values becomes O(2') = (0; 1; 1; 0). Then, when the input values are modified (test step 3), the automaton evolves from location 3 to location 5 and the observed combination of output values becomes O(3) = (0; 0; 1; 1).

Whatever the considered case, the test is negative: the observed output sequence is different from the expected one.

7. Conclusions and perspectives

Construction of a formal model that can be used to build a test sequence for conformance test of a PLC requires to consider not only the specification model of the

logic control, in a tailored-made language, but also the execution mode of this model by the PLC to test. This paper has proposed a method to construct this formal model for two execution modes and in particular for the most common mode in industry, without stability search; then, it has been shown, on the basis of a simple example, that the consideration of the execution mode is mandatory to obtain valid test results.

On-going works are aiming to remove the hypotheses introduced in this study; in particular, extension to models that include time-dependent elements is under investigation. As Mealy machines are not appropriate to model timed systems, this extension requires to select a class of timed automata to represent the formal models.

Furthermore, the validation process in the lifecycle of an automated system does not focus on the behavior of the PLC in isolation but on the behavior of the PLC connected to the plant to form a closed-loop system. This system can be modeled by an automaton whose state space is a subset of the state space of the automaton which represents the isolated PLC, because some combinations of input/output variables are not feasible in the plant. It is thus not necessary to explore the whole state space of the PLC model to validate the closed-loop system. Hence, a promising perspective for further research is to develop formal techniques for validation of PLCs that are complementary to conformance test and rely on the observation of the input/output sequences in the closed-loop system.

References

Bierel E., Douchin O., Lhoste P. (1997, May). Grafcet : from theory to implementation. *Journal Européen des Systèmes Automatisés*, Vol. 31, No. 3, pp. 543-559.

David R., Alla H. (1992). *Petri nets and grafcet - tools for modelling discrete event systems*. Prentice Hall.

IEC 1012. (2004). *Ieee standard for software verification and validation*. Institute of Electrical and Electronics Engineers.

IEC 60848. (2002). *Grafcet specification language for sequential function charts (2nd ed.)*. International Electrotechnical Commission.

Lee D., Yannakakis M. (1996, August). Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, Vol. 84, No. 8, pp. pp. 1090–1123.

Lesage J.-J., Roussel J.-M. (1993, March). Hierarchical approach to grafcet using forcing order. *Automatique Productique Informatique Industrielle*, Vol. 27, No. 1, pp. 25-38.

Lhoste P., Panetto H., Roesch M. (1993). Grafcet : from syntax to semantics. *Automatique Productique Informatique Industrielle*, Vol. 27, No. 1, pp. 127-141. (Special issue "Advances in Grafcet", ISSN 0296-1598)

Naito S., Tsunoyama M., Nagaoka N. (1981). Fault detection by sequential machines by transition-tours. In *Digest of papers: Ftcs-11: the eleventh annual international symposium on fault-tolerant computing, june 24-26, 1981, portland, maine*, p. 238.

Provost J., Roussel J.-M., Faure J.-M. (2011a, August). A formal semantics for Grafcet specifications. In *7th IEEE Conference on Automation Science and Engineering (IEEE CASE 2011)*, p. 488-494. Trieste, Italia.

Provost J., Roussel J.-M., Faure J.-M. (2011b, September). Translating Grafcet specifications into Mealy machines for conformance test purposes. *Control Engineering Practice*, Vol. 19, No. 9, pp. pp. 947–957.