# Test exhaustif de contrôleurs logiques spécifiés en Grafcet : apports et limites d'une modélisation par machines de Mealy

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

# Test exhaustif de contrôleurs logiques spécifiés en Grafcet : apports et limites d'une modélisation par machines de Mealy

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

*Laboratoire Universitaire de Recherche en Production Automatisée*
*École Normale Supérieure de Cachan*
*61, avenue du Président Wilson*
*F-94 235 Cachan Cedex*

*{julien.provost, jean-marc.roussel, jean-marc.faure}@lurpa.ens-cachan.fr*

RÉSUMÉ. *Utilisés pour la commande des systèmes critiques, les contrôleurs logiques doivent faire l'objet de tests rigoureux pour garantir le respect du fonctionnement attendu. Dans ce but, nous proposons une méthode spécifique permettant un test de conformité exhaustif vis-à-vis de spécifications en Grafcet. Cette méthode s'appuie sur les travaux de la communauté informatique relatifs au test à partir de machines de Mealy. La mise en œuvre sur des systèmes réels a mis en évidence certaines limites de telles approches pour le test de contrôleurs logiques.*

ABSTRACT. *Logic controllers can be used for control of critical systems; in this case, they must be rigorously tested to ensure that they behave as specified. To meet this objective, this paper proposes a specific method to obtain an exhaustive conformance test for specifications given in SFC. This method relies on results of the computer science community on test of Mealy machines. Implementation on real systems has revealed some limitations of such approaches in the case of logic controllers, however.*

MOTS-CLÉS : *Test de conformité, Test piloté par le modèle, Grafcet, Machine de Mealy, Contrôleurs logiques.*

KEYWORDS: *Conformance test, Model-based test, SFC, Mealy machine, Logic controllers.*

JESA – 43/2009. MSR 2009, pages 889 à 904

## 1. Introduction

Les systèmes critiques sont aujourd'hui de plus en plus souvent commandés par des contrôleurs logiques, tels que les calculateurs embarqués pour l'automobile ou les API (Automates Programmables Industriels) pour le transport ferroviaire et les centrales électriques. Afin de garantir le niveau de sûreté exigé pour ce type de systèmes, le comportement de ces contrôleurs doit être testé le plus rigoureusement possible. Proposer des méthodes de test de ces équipements est l'objectif du projet de recherche TESTEC (TEst de Systèmes Temps réel Embarqués Critiques) financé par l'ANR, dont les partenaires industriels sont EDF R&D et Geensys. Dans le cadre de ce projet, le LURPA a en charge la définition d'une procédure de test de conformité exhaustif et non-invasif pour les systèmes logiques hautement critiques. Le test de conformité d'un contrôleur logique consiste à (cf. figure 1) :

– solliciter le contrôleur à tester par une séquence d'entrée pré-établie ;

– observer sa réponse à cette séquence ;

– comparer la séquence de sortie observée à la séquence de sortie attendue.

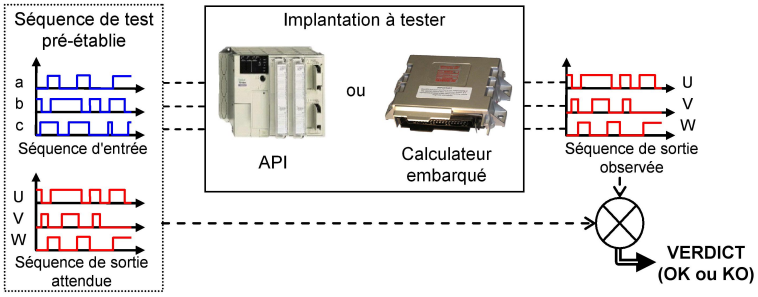


**Figure 1.** *Principe du test de conformité*

La séquence de test sur laquelle repose le test de conformité est donc composée d'une séquence d'entrée et d'une séquence de sortie attendue. Dans la pratique industrielle actuelle, ces séquences sont généralement obtenues de manière manuelle nécessitant un travail fastidieux et malheureusement source d'erreurs.

Le test de conformité d'une réalisation par rapport à sa spécification est pourtant un sujet de recherche pour lequel de nombreux résultats existent. Une synthèse de ces travaux, issus de la communauté informatique, peut être trouvée dans (Jeron, 2004). Leur but est de construire automatiquement une séquence de test à partir d'une spécification exprimée dans un langage formel tel que les machines de Mealy ou les systèmes de transitions (Brinksma *et al.*, 1990), (Lee *et al.*, 1996), (Tretmans, 2008).

Toutefois, dans la limite de nos connaissances, aucun de ces travaux n'a abordé la question de la construction de séquence de test à partir de spécifications exprimées à l'aide de modèles en langage normalisé conçus pour le contrôle/commande. L'objectif

de notre travail est de combler cette lacune lorsque la commande est spécifiée en Grafcet (IEC60848, 2002). Une méthode permettant d'obtenir, à partir d'un grafcet décrivant le comportement attendu d'un contrôleur logique, un test exhaustif validant la conformité du comportement du contrôleur en fonctionnement avec sa spécification a donc été définie (cf. figure 2). La séquence de test utilisée est conçue pour couvrir la totalité de l'espace d'état défini par la spécification afin de garantir à 100 % la conformité du comportement.

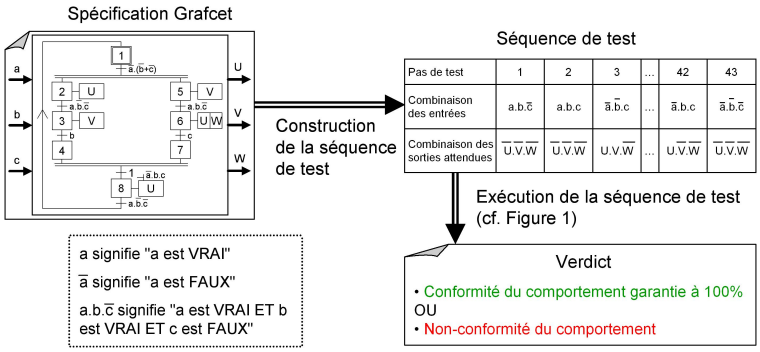


**Figure 2.** *Objectif de ces travaux*

Les différentes phases de notre approche sont présentées dans la section 2. La section 3 est consacrée à la représentation d'un comportement spécifié en Grafcet à l'aide d'une machine de Mealy. La section 4 est dédiée à la mise en œuvre effective sur un contrôleur logique en fonctionnement d'un test de conformité basé sur une séquence de test d'une machine de Mealy. Cette dernière phase a permis de mettre en évidence certaines limites de la séquence de test déduite d'une machine de Mealy pour le test de contrôleurs logiques, limites que nous détaillerons et discuterons.

## 2. Présentation globale de la méthode

Pour fournir des résultats fiables, le test de conformité d'un contrôleur pilotant un système hautement critique doit :

– **Être effectué en boîte noire**. La structure interne du contrôleur n'est pas connue. Son comportement ne peut être déterminé que par l'observation de ses réponses aux sollicitations d'entrée.

– **Être non invasif**. Aucune sonde ni portion de code ne peut être introduite dans le contrôleur. Il est donc impossible d'obtenir les valeurs des variables internes.

– **Être exhaustif**. La totalité de l'espace d'état de la spécification doit être explorée. Dans la suite de cette communication, la taille de cet espace d'état sera supposée être telle que son parcours exhaustif soit possible. Cette hypothèse est tout à fait raisonnable dans le cas des fonctions de sécurité de systèmes hautement critiques qui

comportent généralement peu d'entrées / sorties. La question du passage à l'échelle ne sera donc pas abordée dans cette communication.

Pour tester de manière exhaustive le comportement d'un contrôleur, il est nécessaire de :

– construire la totalité de l'espace d'état décrit dans la spécification ;

– trouver une stratégie permettant de parcourir totalement cet espace d'état ;

– solliciter le contrôleur suivant la stratégie retenue ;

– analyser l'adéquation entre la réponse du contrôleur et celle attendue.

Quand la spécification est donnée en Grafcet (cf. figure 3), il est important de noter les points suivants :

– L'espace d'état de la spécification n'est pas défini explicitement. Un état correspond à la combinaison de plusieurs étapes actives simultanément, une évolution peut correspondre au franchissement simultané de plusieurs transitions (*Problème 1*).

– Dans un grafcet, une transition peut être franchissable pour plusieurs combinaisons différentes des entrées. Pour d'autres combinaisons de ces entrées, aucune transition ne doit être franchie (*Problème 2*).

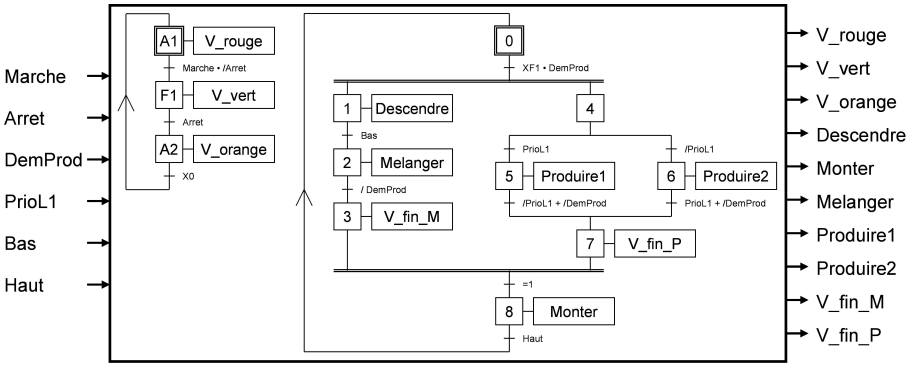


**Figure 3.** *Exemple de spécification en Grafcet à l'origine d'un test de conformité*

Construire la totalité de l'espace d'état d'un grafcet pour décrire de manière exhaustive son comportement pose une réelle difficulté dans le cas général. Dans cette communication, **seuls les systèmes non temporisés sont étudiés**. Cela implique que la spécification en Grafcet ne comprenne pas de réceptivités ni d'actions dépendantes du temps. Cette contrainte est la principale limite actuelle de notre approche. Pour autant, aucune contrainte n'est imposée sur la structure du grafcet (séquences en parallèle, séquences concurrentes, rendez-vous...).

Étant donné ces précisions, le test d'un contrôleur spécifié en Grafcet réutilisant les résultats des techniques de test de machines de Mealy nécessite d'effectuer séquentiellement les trois phases suivantes :

1) Représentation du comportement spécifié en Grafcet à l'aide d'une machine de Mealy ;

2) Construction de la séquence de test nécessaire au test de conformité exhaustif de cette machine de Mealy ;

3) Réalisation du test de conformité à partir de la séquence obtenue.

La deuxième phase de notre approche ne sera pas détaillée dans cette communication. L'obtention de la séquence de test à partir de la machine de Mealy décrivant le comportement spécifié en Grafcet repose sur la théorie des graphes puisqu'elle consiste à construire un circuit fermé traversant chaque arc de la machine de Mealy au moins une fois. Nous avons retenu la méthode du tour de transition (Naito *et al.*, 1981) pour sa bonne adéquation à notre besoin (Provost *et al.*, 2009).

## 3. Représentation d'un comportement spécifié en Grafcet à l'aide d'une machine de Mealy

Cette section rappelle tout d'abord le principe du test de conformité de machines de Mealy, ce qui permet de montrer l'intérêt d'un tel modèle pour le test de conformité. L'obtention d'une machine de Mealy décrivant un comportement spécifié en Grafcet est ensuite détaillée.

### 3.1. *Principe du test de conformité d'une machine de Mealy*

Le problème du test de conformité d'une machine de Mealy peut être énoncé de la manière suivante : soit $S$ une machine de Mealy connue (la spécification) et $I$ une machine de Mealy inconnue (l'implantation à tester) dont on peut seulement observer les entrées / sorties ; déterminer par un test (une séquence finie d'entrées / sorties) si $I$ est équivalente à $S$.

Pour que ce problème admette une solution, il est généralement supposé que la spécification est minimale et fortement connexe. L'équivalence entre une implantation $I$ et une spécification $S$ revient alors à vérifier qu'aucune des fautes suivantes n'apparaisse lors du test de $I$ (cf. figure 4) :

– Faute de sortie : depuis un état $s$, pour une entrée $i$, $I$ produit la sortie $o'$ au lieu de la sortie attendue $o$.

– Faute de transfert : depuis un état donné $s$, après franchissement de la transition étiquetée $i/o$, l'état d'arrivée est $s''$ au lieu de $s'$.

La séquence de test de $I$, construite à partir de $S$, doit permettre de détecter ces deux types de fautes, pour chaque état et chaque transition.

Pour tirer parti des travaux réalisés sur les machines de Mealy, il est nécessaire de savoir exprimer le comportement d'une spécification Grafcet à l'aide d'une machine de Mealy. Cette activité est effectuée en deux temps :

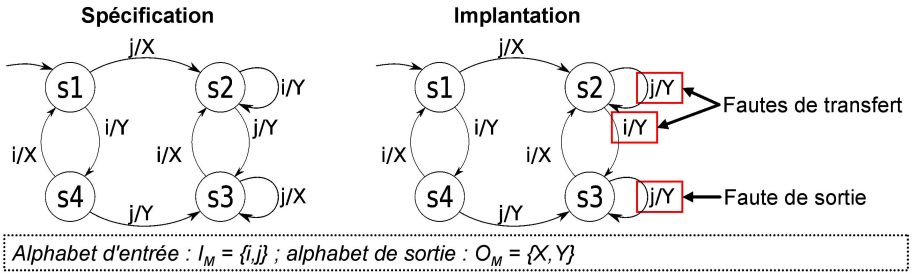


**Figure 4.** *Fautes de transfert et de sortie dans une machine de Mealy*

1) Construction du graphe des situations accessibles qui représente le comportement du grafcet initial.

2) Transcription de ce graphe, doté de conditions de transition qui sont des expressions booléennes, en une machine de Mealy.

Le premier point apporte une solution au *Problème 1*, le second au *Problème 2* définis à la section 2.

### 3.2. *Construction du graphe des situations accessibles (GSA)*

Les lecteurs intéressés pourront trouver dans (Roussel *et al.*, 1996) une présentation détaillée de cette construction. Nous rappellerons seulement ici les principales caractéristiques du GSA obtenu.

– Tout grafcet ne comportant pas d'élément temporisé (réceptivité ou action) peut être transcrit en un Graphe des Situations Accessibles décrivant toutes les évolutions possibles de ce grafcet. Chaque situation du GSA représente un état du grafcet, c'est-à-dire l'ensemble des étapes actives à un instant donné.

– Un GSA ne contient qu'une situation active à la fois, correspondant à la situation courante du grafcet.

– Chaque situation d'un GSA est atteignable depuis la situation initiale.

– Les conditions de transition d'un GSA sont des expressions booléennes formées à partir des seules entrées du grafcet initial.

– Les valeurs des sorties d'un GSA ne dépendent que de la situation active.

– Un GSA ne contient pas d'évolution fugace (une définition formelle est donnée équation [6]). Les éventuelles évolutions fugaces présentes dans le grafcet initial sont transformées en évolutions stables lors de la génération du GSA.

Pour permettre un test non invasif, nous supposerons qu'à chaque situation du GSA est associé un ensemble différent d'actions, permettant ainsi la connaissance

de la situation active uniquement par observation des sorties émises. Il convient de signaler que cette contrainte ne constitue pas une limite théorique pour l'application de notre approche, car il est toujours possible d'ajouter des sorties complémentaires utilisées uniquement lors de la phase de test.

### 3.3. *Transcription d'un GSA en une machine de Mealy*

Cette section vise à définir les règles de transcription d'un GSA en une machine de Mealy équivalente. Il est important de rappeler qu'aux transitions d'un GSA sont associées des expressions booléennes (cf. figure 5). A l'inverse, une machine de Mealy est un modèle à base d'événements. Par conséquent, le problème scientifique posé est la transcription d'une machine à états dont les conditions de transition sont définies par des expressions booléennes en une machine à base d'évènements, cela sans perte de sémantique.

#### 3.3.1. *Définition formelle d'un Graphe des Situations Accessibles*

Un GSA est un 6-uplet : $GSA = (V_I, V_O, S_{GSA}, s_{initGSA}, T, A)$, où :

– $V_I$ est l'ensemble non vide des variables d'entrée, (Cardinal de $V_I$ : $|V_I| = n_{V_I}$).

– $V_O$ est l'ensemble non vide des variables de sortie, ($|V_O| = n_{V_O}$).

– $S_{GSA}$ est l'ensemble des situations, ($|S_{GSA}| = n_{S_{GSA}}$).

– $s_{initGSA} \in S_{GSA}$ est la situation initiale.

– $T$ est l'ensemble des transitions $t$.

– $A$ est l'ensemble des actions $a$.

Par construction, $V_I$, $V_O$ et $S_{GSA}$ sont des ensembles distincts non vides.

Une transition $t$ de $T$ est définie par le triplet : $t = (Am(t), Av(t), R(t))$, où :

– $Am(t)$ est la situation en amont de la transition $t$, ($Am(t) \in S_{GSA}$).

– $Av(t)$ est la situation en aval de la transition $t$, ($Av(t) \in S_{GSA}$)

– $R(t)$ est la réceptivité associée à la transition $t$. Une réceptivité $R(t)$ est une expression booléenne formée à partir des seules variables d'entrée $V_I$.

Une action $a$ de $A$ est définie par le couple : $a = (s(a), o(a))$, où :

– $s(a)$ est la situation à laquelle l'action $a$ est associée, ($s(a) \in S_{GSA}$).

– $o(a)$ est la sortie sur laquelle influe $a$, ($o(a) \in V_O$).

Par construction, le GSA est bien structuré, c'est-à-dire que :

– Tout franchissement d'une transition correspond à un changement de situation :

$$\forall t \in T \quad [Am(t) \neq Av(t)] \qquad [1]$$

– Les transitions ne sont pas redondantes :

$$\forall (t_i, t_j) \in T^2 \quad [Am(t_i) = Am(t_j) \Rightarrow Av(t_i) \neq Av(t_j)] \qquad [2]$$

– Les actions ne sont pas redondantes :

$$\forall (a_i, a_j) \in A^2 \quad [s(a_i) = s(a_j) \Rightarrow o(a_i) \neq o(a_j)] \qquad [3]$$

– Toute situation du GSA est atteignable depuis la situation initiale ; il existe une séquence finie de transitions depuis la situation initiale vers chaque situation du GSA :

$$\begin{array}{l} \forall s \in \{S_{GSA} - \{s_{initGSA}\}\} \ \exists (t_1..t_n) \in T^n (n \in \mathbb{N}^*) \\ \left[ \left\{ \begin{array}{l} Am(t_1) = s_{initGSA} \\ Av(t_n) = s \\ \forall i \in [1, n-1], Av(t_i) = Am(t_{i+1}) \end{array} \right. \right] \end{array} \qquad [4]$$

– Les transitions en aval d'une même situation d'un GSA sont deux à deux exclusives (structure de graphe d'état) :

$$\forall s \in S_{GSA} \ \forall (t_i, t_j) \in T^2 \quad [Am(t_i) = Am(t_j) = s \Rightarrow R(t_i) \cdot R(t_j) = 0] \qquad [5]$$

– Le GSA ne contient pas d'évolution fugace :

$$\forall s \in S_{GSA} \ \forall (t_i, t_j) \in T^2 \quad [Av(t_i) = Am(t_j) = s \Rightarrow R(t_i) \cdot R(t_j) = 0] \qquad [6]$$

Pour pouvoir tester de manière non-invasive ce GSA, il est nécessaire de pouvoir déterminer quelle est la situation active par la seule observation de ces sorties. Pour cela, à chaque situation doit être associé un ensemble différent d'actions :

$$\forall (s_i, s_j) \in S_{GSA}^2 \quad [O(s_i) \neq O(s_j)] \qquad [7]$$

où $O(s_i)$ est le sous-ensemble de $V_O$ regroupant toutes les sorties émises lorsque la situation $s_i$ est active.

$$O(s_i) = \{o(a) \mid \exists a \in A \quad [s(a) = s_i]\} \qquad [8]$$

Pour l'exemple présenté figure 3, le GSA est composé de 16 situations, 102 transitions et 42 actions.

### 3.3.2. *Construction de la machine de Mealy*

Nous proposons que la machine de Mealy $M$ : ($I_M$,$O_M$,$S_M$,$s_{initM}$,$\delta_M$,$\lambda_M$), représentant le GSA : ($V_I$,$V_O$,$S_{GSA}$,$s_{initGSA}$,$T$,$A$), soit définie comme suit :

– $I_M$ : alphabet d'entrée. Cet alphabet contient $2^{n_{V_I}}$ éléments. Chaque élément $i_i$ de cet alphabet représente une combinaison distincte des variables logiques de $V_I$. A chaque élément $i_i$ est associé un unique minterme[1] formé à partir des variables de $V_I$ représentant cette combinaison distincte. Par construction, nous avons :

$$\forall (i_i, i_j) \in I_M^2 \quad [m_I(i_i) \cdot m_I(i_j) = 0] \qquad [9]$$

1. Un minterme est une expression logique des $n$ variables utilisant uniquement l'opérateur logique ET ($\cdot$) et l'opérateur complément ($^-$).

Par exemple, $m_I(i_{16}) = \overline{Arret} \cdot Bas \cdot \overline{DemProd} \cdot \overline{Haut} \cdot \overline{Marche} \cdot \overline{PrioL1}$

– $O_M$ : alphabet de sortie. Cet alphabet contient $2^{nv_O}$ éléments. Chaque élément $o_i$ de cet alphabet représente une combinaison distincte des variables logiques de $V_O$. A chaque élément $o_i$ est associé un unique minterme formé à partir des variables de $V_O$ représentant cette combinaison distincte.

– $S_M$ : ensemble des états. A toute situation du GSA est associée un état de la machine de Mealy $M$. Dans un souci de lisibilité, l'état associé à une situation et cette situation seront notés de manière identique. Ainsi, l'ensemble $S_M$ est identique à l'ensemble $S_{GSA}$.

– $S_{initM}$ : état initial. Cet état est associé à la situation initiale du GSA.

– Les fonctions de transition $\delta_M$ et de sortie $\lambda_M$ caractérisant la machine de Mealy sont définies à partir des éléments de $V_I$, $V_O$, $S_{GSA}$, $T$, $A$ et $I_M$, $O_M$, $S_M$. Elles doivent être telles que la machine de Mealy recherchée soit déterministe et complètement spécifiée, c'est-à-dire [2] :

$$\forall (s,i) \in S_M \times I_M \quad \left[ \begin{array}{l} \exists ! \delta_M(s,i) \in S_M \\ \exists ! \lambda_M(s,i) \in O_M \end{array} \right] \qquad [10]$$

La fonction de transition $\delta_M$ de la machine de Mealy est définie par :

$$\forall s \in S_M \ \forall i \in I_M \quad \left[ \begin{array}{l} \text{si } \exists t \in T \left[ \begin{array}{l} Am(t) = s \\ R(t) \cdot m_I(i) = m_I(i) \end{array} \right] \\ \quad \text{alors } [\delta_M(s,i) = Av(t)] \\ \text{sinon} \\ \quad \text{alors } [\delta_M(s,i) = s] \end{array} \right] \qquad [11]$$

La fonction de transition proposée $\delta_M$ est bien complètement spécifiée puisqu'une valeur est définie pour chaque couple $(s,i) \in S_M \times I_M$. Cette valeur est unique car les transitions en aval d'une situation $s$ sont deux à deux exclusives (cf. équation [5]).

La construction de la fonction de sortie $\lambda_M$ s'appuie sur le fait que les sorties émises ne dépendent que de la situation active du GSA. La fonction de sortie $\lambda_M$ est définie par :

$$\begin{array}{l} \forall s \in S_M \ \forall i \in I_M \quad [\lambda_M(s,i) = o_j] \text{ tel que :} \\ \left[ \begin{array}{ll} \forall v \in O(\delta_M(s,i)) & [m_O(o_j) \cdot v = m_O(o_j)] \\ \forall v \in \{V_O - O(\delta_M(s,i))\} & [m_O(o_j) \cdot \bar{v} = m_O(o_j)] \end{array} \right] \end{array} \qquad [12]$$

Les définitions de $\delta_M$ et $\lambda_M$ proposées assurent le déterminisme d'évolution et d'émission des sorties. Chaque état de cette machine de Mealy est atteignable depuis l'état initial puisque cette contrainte est vérifiée pour le GSA (cf. équation [4]).

2. $\exists !$ : Il existe un unique élément.

De plus, d'après les définitions des fonctions $\delta_M(s,i)$ et $\lambda_M(s,i)$, chaque couple $(s,i) \in S_M \times I_M$ vérifie :

$$\begin{array}{l} \forall((s,i),(s',i')) \in (S_M \times I_M)^2 \\ [\delta_M(s,i) = \delta_M(s',i') \Rightarrow \lambda_M(s,i) = \lambda_M(s',i')] \end{array} \quad [13]$$

Ce résultat est la conséquence de deux propriétés du GSA (les valeurs de sorties ne dépendent que de la situation active ; à chaque situation est associée un ensemble distinct d'actions) et assure la minimalité de la machine de Mealy obtenue. **Cette machine de Mealy peut donc être la source d'un test de conformité exhaustif**.

La machine de Mealy ainsi construite contient autant d'états qu'il y a de situations dans le GSA. Le nombre de ses transitions ne dépend que du nombre de situations et de variables d'entrée du GSA.

$$\left\{ \begin{array}{l} n_{etats} = n_{S_{GSA}} \\ n_{transitions} = n_{S_{GSA}} \cdot 2^{n_{V_I}} \end{array} \right. \quad [14]$$

Il est important de noter que le nombre de transitions du GSA n'a aucune influence sur la taille de la machine de Mealy obtenue. Pour la spécification proposée figure 3, la machine de Mealy obtenue contient 16 états et 1024 transitions. La figure 5 présente une portion du GSA ainsi qu'une portion de la machine de Mealy obtenue à partir de cet exemple.

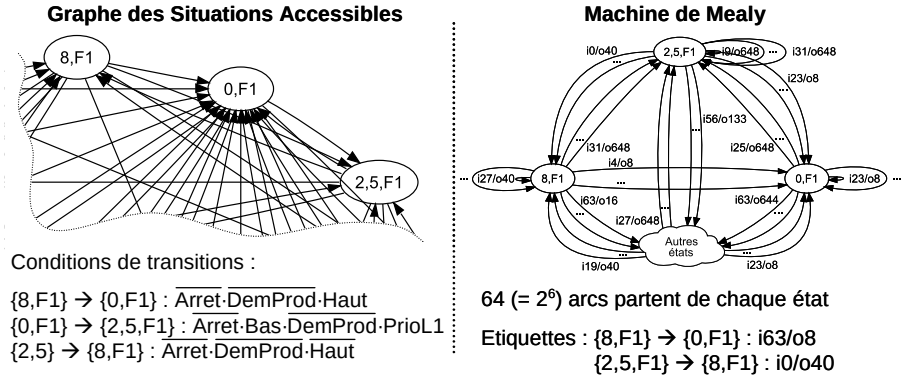


**Figure 5.** *Portions du GSA et de la machine de Mealy pour l'exemple de la figure 3*

Pour cet exemple, la séquence de test obtenue par la méthode du tour de transition (Naito *et al.*, 1981), à partir de la machine de Mealy équivalente, comporte 1966 pas.

## 4. Mise en œuvre d'un test de conformité de contrôleurs logiques basé sur une séquence de test d'une machine de Mealy

Afin de pouvoir juger de l'applicabilité de la méthode proposée pour des contrôleurs logiques réels, un banc de test original a été développé. Après une rapide description de ce banc, sa mise en œuvre est décrite, ce qui permet en particulier de fixer l'ordre de grandeur de la durée du test. Des détections erronées de non-conformité (programmes de contrôle conformes détectés non conformes à tort) étant apparues lors de certains tests, la suite de cette section discute l'origine de ces erreurs puis, propose un principe de construction de la séquence de test permettant de les éviter.

### 4.1. *Présentation du banc de test développé*

Le banc de test utilisé pour ces travaux a été conçu à partir des retours d'expérience d'une précédente plate-forme développée au LURPA pour l'identification des systèmes logiques (De Smet *et al.*, 1999) (Denis *et al.*, 2001).

Pour pouvoir tester la conformité d'un contrôleur en fonctionnement par la seule observation du comportement de ses sorties à des sollicitations de ses entrées, il est nécessaire de :

– relier physiquement point à point chaque entrée ou sortie logique du contrôleur à tester au banc de test ;

– disposer de moyens matériels pouvant faire varier chacune des entrées logiques du contrôleur en fonction de la séquence de test établie ;

– disposer de moyens permettant d'observer chacune des sorties logiques.

Le banc de test utilisé actuellement est construit autour d'un module d'entrées / sorties déportées. Ce module industriel (Schneider Electric Momentum 170ENT11001) dispose de 16 entrées logiques et 16 sorties logiques contrôlables à distance à partir d'un simple calculateur via le protocole ModBus TCP/IP. Chaque sortie de ce module d'entrées / sorties déportées est connectée en amont d'une entrée du contrôleur à tester tandis que chacune de ses entrées est connectée en aval d'une sortie du contrôleur à tester (cf. figure 6). Cette solution ouverte offre ainsi le maximum de possibilités pour réaliser physiquement les sollicitations et les observations nécessaires au test de conformité.

Concernant le contrôleur à tester, un API modulaire (Phoenix Contact ILC 170 ETH 2TX) a été retenu, en raison de sa bonne réactivité (temps de cycle pouvant être inférieur à 5 ms) et de la conformité de son atelier de programmation à la norme IEC 61131-3 (IEC61131-3, 1993).

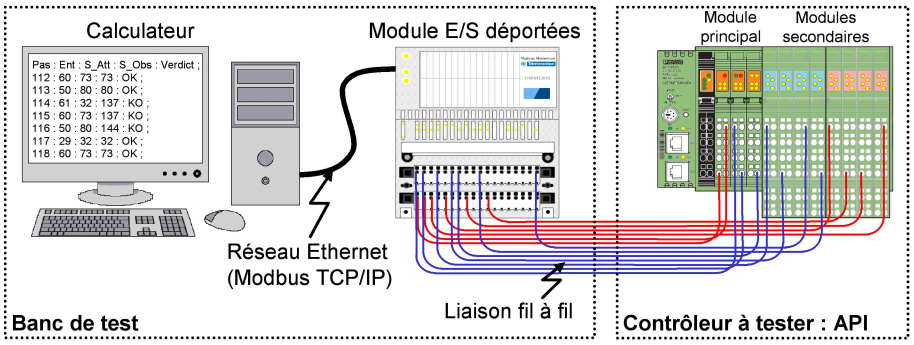


**Figure 6.** *Description du dispositif expérimental*

### 4.2. *Réalisation d'un test de conformité*

Il a été choisi de réaliser le test de conformité du contrôleur dans les conditions les plus proches de son fonctionnement réel. Une fois programmé, le contrôleur à tester est déconnecté de son atelier de programmation pour toute la durée du test. Le code est implanté dans l'API en langage Structured Text obtenu automatiquement à partir de la spécification Grafcet en suivant la méthode proposée dans (Machado *et al.*, 2006). Cette solution nous permet, soit de générer un code conforme à la spécification, soit d'introduire artificiellement des erreurs de programmation.

Le déroulement d'un test de conformité est le suivant :

– Programmation si nécessaire du contrôleur à tester ;

– Connexion physique du contrôleur à tester au banc de test ;

– Redémarrage du contrôleur à tester ;

– Exécution de la séquence de test. Cette exécution consiste, pour chaque pas de la séquence de test, à :

- émettre la combinaison des entrées logiques correspondant à la sollicitation demandée ;

- attendre le délai nécessaire à la stabilisation des sorties émises ;

- lire la combinaison des sorties émises par le contrôleur à tester ;

- comparer les sorties émises aux sorties attendues ;

- enregistrer le résultat.

Pour des raisons de commodité, les premiers essais ont été réalisés avec un API en fonctionnement périodique réglé à 20 ms (le code est exécuté toutes les 20 ms). La lecture des sorties de l'API est faite après un temps d'attente de 200 ms afin de prendre en compte les deux paramètres suivants (cf. figure 7) :

– Retard de causalité minimal ;

– Nombre de pas de calcul maximum pour que les valeurs de toutes les sorties soient stabilisées.

Pour le grafcet proposé sur la figure 3, la séquence comporte 1966 pas conduisant à la réalisation du test de conformité dans son intégralité en moins de 7 minutes.

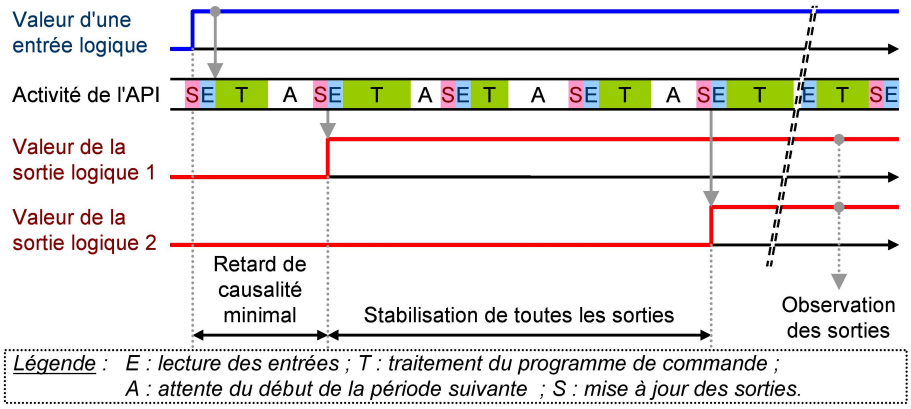


**Figure 7.** *Chronogramme illustrant le temps d'attente avant observation des sorties (cas d'un fonctionnement périodique)*

### 4.3. *Étude de la robustesse de la méthode de test*

Les premiers résultats étant concluants, nous nous sommes alors intéressés à la robustesse de l'approche en étudiant comment influent sur le résultat du test les caractéristiques techniques du contrôleur à tester (mode de fonctionnement (cyclique ou périodique) et répartition des entrées).

Lors de la réalisation du test de conformité du comportement du contrôleur en fonctionnement cyclique (mode de fonctionnement le plus couramment utilisé car maximisant la réactivité de la commande), nous avons relevé des détections erronées de non-conformité conduisant au rejet par le banc de test d'un programme conforme. Une analyse détaillée des fichiers de résultats a permis de montrer que ces problèmes sont induits par des erreurs de perception par le contrôleur testé de la sollicitation physique générée par le banc de test (cf. figure 8).

En effet, l'étude théorique qui a été conduite repose sur l'hypothèse implicite que toute sollicitation émise par le banc de test est perçue par le contrôleur à tester sans la moindre erreur d'interprétation. Dans la pratique, il s'avère que cette hypothèse peut ne pas être vérifiée lorsque deux sollicitations successives nécessitent de faire varier simultanément plusieurs entrées logiques.

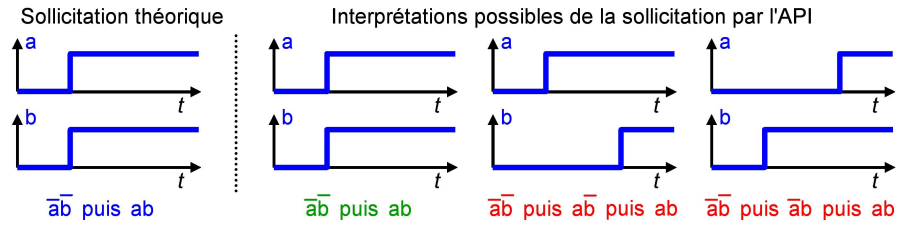


**Figure 8.** *Interprétations possibles d'une variation simultanée de 2 entrées logiques*

Pour analyser la sensibilité du contrôleur à tester à cet aléa d'interprétation, l'expérimentation suivante a été conduite :

– Le banc de test sollicite le contrôleur en faisant varier simultanément et de manière identique (passage de 0 à 1, puis de 1 à 0) 8 de ses entrées.

– Lorsque le contrôleur détecte le premier passage à un ou à zéro d'une de ses entrées, il recopie la valeur courante de chaque entrée sur une sortie : la valeur du vecteur de sortie est donc égale à la valeur du vecteur d'entrée à l'instant où le premier changement est détecté.

– Le banc de test lit la valeur courante des sorties émises et compare cette valeur à la valeur attendue des sorties pour la sollicitation correspondante. Un compteur est incrémenté lorsqu'une erreur est détectée sur l'une des voies.

Le tableau 4.3 regroupe les taux d'erreur d'interprétation des sollicitations par l'API testé en fonction de son mode de fonctionnement et de la répartition des entrées sur ses composants internes (toutes regroupées sur le module principal, toutes regroupées sur le module secondaire, réparties entre le module principal et le module secondaire). Ces chiffres ont été établis à partir de campagnes comportant chacune 200 000 essais.

| Répartition \ Mode | | Cyclique | Périodique 10ms | Périodique 20ms |
|---|---|---|---|---|
| sur module principal | Passage de 0 à 1 | 0.062 % | 0.022 % | 0.014 % |
| | Passage de 1 à 0 | 1.915 % | 0.886 % | 0.455 % |
| sur module secondaire | Passage de 0 à 1 | 0.104 % | 0.040 % | 0.020 % |
| | Passage de 1 à 0 | 2.536 % | 0.993 % | 0.550 % |
| entre les deux modules | Passage de 0 à 1 | 29.17 % | 39.62 % | 20.23 % |
| | Passage de 1 à 0 | 38.55 % | 43.46 % | 21.89 % |

**Tableau 1.** *Taux d'erreurs d'interprétation par l'API*

Ces erreurs d'interprétation par l'API des sollicitations émises par le banc de test sont dues à l'absence de synchronisation entre le banc et l'API ainsi qu'aux méca-

nismes internes de l'API pour l'acquisition de ses entrées. Lors de cette phase d'acquisition, le moniteur d'exécution interroge en effet de manière asynchrone les entrées physiques pour obtenir les valeurs courantes des entrées logiques. Les erreurs d'interprétation sont alors dues à une interrogation à l'instant exact où ces informations sont construites. Dans le cas d'une répartition des entrées sur plusieurs modules (dernière ligne du tableau 4.3), les erreurs sont augmentées en raison du protocole de communication entre les modules retenu par le constructeur. Ce protocole, conçu pour diminuer les temps d'attente du processeur de l'API, n'est pas prévu pour garantir la simultanéité de lecture des entrées réparties entre plusieurs modules.

### 4.4. *Contraintes induites sur la séquence de test*

Les expériences qui ont été conduites montrent que l'hypothèse implicite selon laquelle toute sollicitation émise par le banc de test est perçue sans la moindre erreur d'interprétation par le contrôleur à tester peut ne pas être vérifiée.

Pour éviter cet aléa, plusieurs stratégies sont envisageables :

– Synchronisation du banc de test avec le contrôleur à tester.

– Réalisation des tests de conformité pour une configuration du contrôleur à tester telle que le taux d'erreurs de perception soit le plus faible possible.

– Exécution à plusieurs reprises du même test de conformité et analyse statistique des résultats.

– Intégration de ce risque d'aléa lors de la constitution d'une séquence de test.

En raison de la forte criticité des systèmes pour lesquels cette méthode doit être mise au point, c'est cette dernière stratégie qui a été retenue. Il est donc nécessaire de s'assurer que la séquence de sollicitations utilisée pour le test de conformité ne puisse pas être à l'origine d'erreur de perception. Une condition suffisante pour garantir ceci est d'imposer qu'une seule entrée logique varie entre deux pas consécutifs de la séquence de test.

Sur le plan théorique, l'ajout de cette contrainte sur la construction de la séquence de test peut conduire, pour certaines spécifications Grafcet, à la non-existence d'une séquence de test garantissant un test exhaustif du comportement sans risque d'aléa.

## 5. Conclusions et perspectives

Les travaux présentés dans cette communication montrent que le test exhaustif d'un contrôleur spécifié en Grafcet est possible sur le plan théorique en s'appuyant sur une formalisation de la sémantique du Grafcet par machine de Mealy. La mise en œuvre effective du test de conformité d'un contrôleur logique par rapport à sa spécification en Grafcet nous a permis d'identifier un risque d'aléa de fonctionnement lorsque la séquence de test est construite sans précaution, c'est-à-dire sans prise en

compte des contraintes technologiques. Ce problème, qu'une seule étude théorique n'aurait pu faire apparaître, peut être résolu en contraignant la constitution de la séquence des sollicitations pour qu'elle ne puisse être à l'origine d'erreurs de perception par le contrôleur testé. Nos travaux en cours portent sur la formalisation d'un critère de testabilité d'une spécification Grafcet, et à l'extension de ces travaux aux systèmes temporisés.

Remerciements

Ce travail a été réalisé dans le cadre du projet TESTEC (ANR TLOG 07-022).

## 6. Bibliographie

Brinksma E., Alderden R., Langerak R., van de Lagemaat J., Tretmans J., « A formal approach to conformance testing », *In J. de Meer, L. Mackert, and W. Effelsberg, editors, Second International Workshop on Protocol Test Systems*, vol. , p. 349-363, 1990.

De Smet O., Roussel J.-M., Hevin N., « Identification de machine séquentielle binaire : application à un système réactif », *2ème congrès sur la modélisation des systèmes réactifs, MSR'99*, p. 351-360, 1999.

Denis B., De Smet O., Lesage J.-J., Roussel J.-M., *Dispositif et procédé d'analyse de performances et d'identification comportementale d'un système en tant qu'automate à événements discrets et finis*, Brevet N° 01 110 933, 2001.

IEC60848, *GRAFCET specification language for sequential function charts*, n° 2, International Electrotechnical Commission, Geneva, Switzerland, 2002.

IEC61131-3, *Programmable controllers - Part 3 : Programming languages*, International Electrotechnical Commission, Geneva, Switzerland, 1993.

Jeron T., Contribution à la génération automatique de tests pour les systèmes réactifs, Habilitation à diriger des recherches, Université de Rennes 1, France, 2004.

Lee D., Yannakakis M., « Principles and methods of testing finite state machines - A survey », *Proceedings of the IEEE*, vol. 84, p. 1090-1123, 1996.

Machado J., Denis B., Lesage J.-J., Faure J.-M., Ferreira Da Silva J., « Logic controllers dependability verification using a plant model », *3rd IFAC Workshop on Discrete-Event System Design, DESDes'06*, p. 37-42, 2006.

Naito S., Tsunoyama M., « Fault Detection for Sequential Machines by Transitions Tours », *Proceedings of the IEEE Fault Tolerant Computer Symposium*, p. 238-243, 1981.

Provost J., Roussel J.-M., Faure J.-M., « Test sequence construction from SFC specification », *Proceedings of 2nd IFAC Workshop on Dependable Control of Discrete Systems (DCDS'09)*, 2009.

Roussel J.-M., Lesage J.-J., « Validation And Verification Of Grafcet Using State Machine », *Proceedings of IMACS-IEEE "CESA'96"*, p. 758-764, 1996.

Tretmans J., « Model Based Testing with Labelled Transition Systems », *in* R. M. Hierons, J. P. Bowen, M. Harman (eds), *Formal Methods and Testing*, vol. 4949 of *Lecture Notes in Computer Science*, Springer, p. 1-38, 2008.