VACSIM

Validation de la commande des systèmes critiques par couplage simulation et méthodes d'analyse formelles

Tâche 3

Apport des techniques d'identification des SED à la validation des systèmes critiques

Livrable L3.1

Construction de modèles formels de contrôleurs logiques pour le test de conformité

Version A

Appel :	**PROGRAMME INGENIERIE NUMERIQUE & SECURITE 2011**
Numéro d'agrément :	**ANR-11-INSE-004**
Thématique:	**Systèmes embarqués et ingénierie du logiciel**
Objectif:	**Apport des techniques d'identification des SED à la validation des systèmes critiques**
Date de démarrage du projet:	**01.10.2011**
Durée :	**42 mois**

Synthèse

Le projet **VACSIM** (Validation de la commande des systèmes critiques par couplage simulation et méthodes d'analyse formelle), référencé ANR-11-INSE-004, étudie les avantages respectifs des techniques de simulation, en incluant des modèles des processus commandés, et des méthodes d'analyse formelle, pour la validation de la commande des systèmes critiques. Ce projet est structuré en 6 tâches.

La tâche 3 «Apport des techniques d'identification des SED à la validation des systèmes critiques», sous la responsabilité du LURPA, a pour objectif de montrer que les techniques d'identification des systèmes à évènements discrets (SED), qui ont été jusqu'à présent principalement développées à des fins de diagnostic à base de modèle, peuvent contribuer à la validation de systèmes critiques lors de leur ingénierie.

Ce livrable L3.1 présente une première contribution à cette tâche s'appuyant sur une communication avec actes présentée aux 5ème Journées Doctorales du GdR MACS (JDMACS'13). Cette contribution présente les résultats des travaux traitant de la validation d'un contrôleur isolé, et non en boucle fermée, et plus précisément de l'influence de certaines hypothèses sur l' algorithme d'implantation de la spécification lors de la construction du modèle formel du contrôleur.

De manière générale, le but de la validation d'un système critique est de garantir que le comportement dudit système est bien identique à celui décrit par sa spécification. De nombreuses méthodes existent pour atteindre cet objectif mais seul le test de conformité d'un contrôleur logique est considéré ici. Cette méthode de validation vise à garantir que ce contrôleur émet les bonnes sorties en réponse à des évènements d'entrée. Pour cela, un scénario, appelé séquence de test, est construit à partir du modèle de la spécification ; cette séquence comporte d'une part une séquence d'évènements d'entrée et d'autre part la séquence d'évènements de sortie attendus en réponse. La séquence d'évènements d'entrée est ensuite appliquée aux bornes du contrôleur et la séquence de sortie émise en réponse est observée et comparée avec celle attendue. Si les deux séquences sont identiques, le comportement du contrôleur peut alors être déclaré conforme à la spécification.

Les travaux présentés ici se situent dans la continuité de ceux réalisés dans le cadre du projet ANR TESTEC (TEst des Systèmes Temps réel Embarqués Critiques – 07 TLOG 022) et plus précisément dans la tâche 5 intitulée «Maîtrise de l'explosion combinatoire et complétude du test pour les systèmes logiques critiques» traitant du test de conformité de contrôleur logique. Ces travaux ont concerné d'une part la construction d'un modèle formel du contrôleur à partir de la connaissance de sa spécification décrite dans un langage normalisé, par exemple le Grafcet, et d'autre part la génération de la séquence de test à partir de ce modèle formel.

Ce livrable se focalise, pour sa part, sur le premier point : construction du modèle formel du contrôleur à partir de sa spécification. En effet, le passage de la spécification au modèle formel du contrôleur implique de prendre en compte l'algorithme d'implantation retenu. Dans le cas de spécifications Grafcet notamment, deux interprétations sont possibles lorsque des évolutions fugaces (plusieurs franchissements successifs d'ensembles de transitions simultanément franchissables possibles sans changement de valeur des variables d'entrée) sont présentes dans la spécification. Dans les travaux réalisés précédemment, l'algorithme d'implantation choisi était celui décrit dans la norme IEC 60848. Celui-ci impose d'attendre d'avoir retrouvé une situation dite stable (aucun franchissement de transition possible sans changement de valeur des variables d'entrée) avant toute mise à jour des variables de sortie et relecture des variables d'entrée.

Bien que ceci soit la manière décrite dans cette norme pour traiter les évolutions fugaces, cette interprétation impose des temps de calcul plus longs qui sont nuisibles à la réactivité du système et peuvent même déclencher le mécanisme de contrôle du temps d'exécution (watchdog) lorsque la séquence de franchissements est trop longue. C'est pourquoi la majorité des contrôleurs industriels (automates programmables ou circuits spécialisés) utilisent un autre algorithme d'implantation, non décrit dans la norme, mais qui assure une mise à jour des variables de sortie plus fréquente.

Le modèle formel du contrôleur obtenu à partir d'une spécification donnée étant fortement dépendant du choix de l'algorithme retenu pour implanter cette spécification, générer le modèle formel qui servira de base à la construction d'une séquence de test selon un autre algorithme que celui réellement utilisé peut bien entendu fausser grandement le résultat du test de conformité réalisé par la suite.

Ce livrable propose donc une méthode formelle de construction de modèle de contrôleur logique à partir d'une spécification Grafcet et de la connaissance de l'algorithme d'implantation de cette spécification. Sont aussi mis en évidence les différences entre les modèles obtenus pour les deux algorithmes d'implantation étudiés, ainsi que l'impact d'un choix erroné sur le verdict du test de conformité.

Un logiciel de type « preuve de concept », nommé TELOCO, a été développé dans le cadre de l'ANR TESTEC afin d'effectuer les opérations de construction de modèle du contrôleur puis de génération de la séquence de test depuis une spécification Grafcet. Ce logiciel a été enrichi, lors des travaux du projet VACSIM, d'une fonctionnalité permettant le choix de l'algorithme d'implantation lors de la construction du modèle formel de contrôleur, afin de générer des séquences de test compatibles avec l'algorithme utilisé pour l'implantation de la spécification dans le contrôleur.

Construction de modèles formels de contrôleurs logiques pour le test de conformité

Anaïs Guignard, Jean-Marc Faure

LURPA, Ecole Normale Supérieure de Cachan
61 Av. Du président Wilson, 94230 Cachan
France

anais.guignard@lurpa.ens-cachan.fr, jean-marc.faure@lurpa.ens-cachan.fr

Résumé— **Ce papier s'intéresse à la construction du modèle formel servant de référence pour le test de conformité d'un contrôleur logique, en supposant que ce contrôleur exécute une spécification Grafcet. Deux algorithmes pour construire ce modèle sont proposés et discutés, selon que l'exécution de la spécification intègre ou non la recherche de situation stable.**

Mots-clés— **test de conformité, Grafcet, automate des localités, stabilité, évolution fugace.**

I. Introduction

L'utilisation croissante de contrôleurs logiques pour réaliser les fonctions de production et de sûreté dans des systèmes critiques requiert de s'assurer, avant la mise en service, que le comportement de chaque contrôleur est conforme à ce qui a été spécifié. Une technique communément employée pour ce faire est le test de conformité [2]. Le principe de cette technique, présenté figure 1, consiste à considérer le contrôleur logique comme une boîte noire, à lui appliquer une séquence d'entrée et à observer si la séquence de sortie observée en réponse correspond à celle prévue par la spécification. L'ensemble constitué de la séquence d'entrée et de la séquence de sortie attendue est dénommé séquence de test.

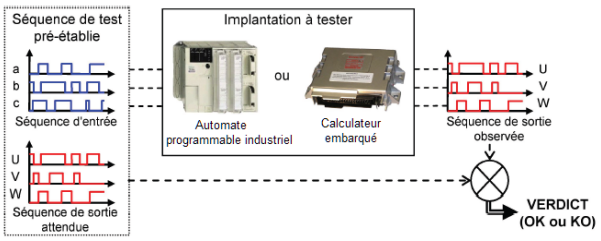

Fig. 1. Principe du test de conformité

Dans le cadre de ces travaux, nous considérons que la spécification est exprimée en Grafcet [3], langage de spécification normalisé et industriellement reconnu par son aspect graphique ainsi que pour ses possibilités de modélisation de comportements complexes (parallélisme, synchronisation, partage de ressources, traitement de sécurités,...). La référence [6] propose une méthode de génération de séquence de test à partir d'une spécification Grafcet, en s'appuyant sur une traduction du Grafcet en machine de Mealy et sur les résultats antérieurs sur le test de conformité des machines de Mealy [4].

Cette méthode suppose que le contrôleur exécute le Grafcet selon un algorithme avec recherche de stabilité, c.a.d. n'émet ses sorties que si la situation du Grafcet est stable pour la valeur courante des entrées. Dans la pratique industrielle cependant, un contrôleur logique exécute le plus souvent le Grafcet sans recherche de stabilité, afin de garantir un temps de réponse plus court et donc une actualisation des sorties plus fréquente. L'objectif de ce papier est donc de proposer une méthode plus générale, permettant de construire le modèle formel d'un contrôleur logique exécutant une spécification Grafcet quel que soit le mode d'exécution choisi.

Les sections II et III rappellent respectivement le modèle formel du Grafcet retenu et les deux modes d'exécution possibles d'un Grafcet pour un contrôleur logique à scrutation cyclique des entrées. La construction du modèle formel de ce contrôleur fait l'objet de la section IV ; les résultats proposés sont illustrés sur un exemple de la littérature en section V. La comparaison des deux modèles formels alors obtenus, présentée section VI, permet de montrer qu'il est indispensable de connaître le mode d'exécution pour obtenir des résultats de test de conformité corrects. Enfin, une conclusion ainsi que des pistes de recherches sont proposées section VII.

II. Grafcet : Modèle de spécification

A. La norme Grafcet IEC 60848

Le Grafcet est un langage de spécification visant à décrire le comportement synchrone souhaité d'un système logique. Un Grafcet est représenté par un graphe ou un ensemble de graphes composés d'étapes et de transitions. Une étape, représentée par un carré, définit un état partiel du système. Une transition, représentée pas un trait horizontal, définit une évolution partielle du système. Une étape ne peut être reliée qu'à des transitions, et une transition ne peut être reliée qu'à des étapes. Ces liaisons sont orientées et représentées par des arcs reliant le bas de l'élément origine à la partie haute de l'élément destination. Lorsque cette liaison est orientée vers le haut, une flèche indiquant le sens de la liaison doit être ajoutée.

Une étape peut être active ou inactive ; une variable booléenne dite d'activité d'étape est donc associée à chaque étape. De plus, à chaque étape, il est possible d'associer une ou plusieurs actions, qui agissent sur les variables internes

ou de sortie. L'ensemble des étapes actives du Grafcet à un instant donné s'appelle la situation du Grafcet. D'autre part, une expression booléenne, appelée réceptivité et exprimant une condition de franchissement, doit être associée à chaque transition. Une receptivité se définit à partir de variables d'entrée, de variables d'activité d'étape ou encore de conditions temporisées.

A partir de cette structure, dont la définition formelle est donnée sous-section II-B, le modèle évolue selon un ensemble de cinq règles définies ainsi [3] :

- Règle n°1 : *La situation initiale, choisie par le concepteur, est la situation à l'état initial.* Elle définit l'ensemble des étapes qui sont actives à l'état initial, toutes les autres étant inactives.
- Règle n°2 : *Une transition est dite validée lorsque toutes les étapes immédiatement précédentes reliées à cette transition sont actives. Le franchissement d'une transition se produit :*
  - *lorsque la transition est validée*
  - *et que la réceptivité associée à cette transition est vraie.*

  Lorsque le franchissement d'une transition est possible, alors celle-ci est obligatoirement franchie.
- Règle n°3 : *Le franchissement d'une transition entraîne simultanément l'activation de toutes les étapes immédiatement suivantes et la désactivation de toutes les étapes immédiatement précédentes.*
- Règle n°4 : *Plusieurs transitions simultanément franchissables sont simultanément franchies.*
- Règle n°5 : *Si, au cours du fonctionnement, une étape active est simultanément activée et désactivée, alors elle reste active.*

La norme définit ainsi de manière textuelle le comportement du modèle de spécification mais aucunement celui du contrôleur qui exécute cette spécification. Pour obtenir un modèle formel du contrôleur, il nous faut tout d'abord proposer une définition formelle du Grafcet.

B. Modèle formel de la spécification

Dans le cadre de ces travaux, plusieurs hypothèses sur la spécification sont formulées :

- aucun élément du Grafcet n'est temporisé,
- aucune réceptivité ne comporte de condition sur le changement d'état (front) d'une variable logique,
- le Grafcet ne comporte ni encapsulation, ni macro-étape, ni forçage,
- le Grafcet ne comporte ni étape source, ni étape puits.

Sous ces hypothèses, un Grafcet se définit formellement comme un $6-uplet$ $(I_G, O_G, S_G, T_G, A_G, S_{Init})$ où :

- I_G est l'ensemble fini non vide des entrées du système logique,
- O_G est l'ensemble fini non vide des sorties du système logique,
- S_G est l'ensemble non vide des étapes du Grafcet, avec X_G l'ensemble de variables d'activité associées,
- T_G est l'ensemble non vide des transitions du Grafcet,
- A_G est l'ensemble des actions du Grafcet,
- S_{Init} est l'ensemble des étapes initiales du Gracet.

Une transition $t \in T_G$ est définie par un $3-uplet$ $(S_U, S_D, FC(I_G, X_G))$ où :

- S_U est l'ensemble non vide des étapes amont,
- S_D est l'ensemble non vide des étapes aval,
- $FC(I_G, X_G)$ est la réceptivité associée à la transition, expression booléenne dépendant des variables d'entrée et des variables d'activité d'étape.

L'ensemble des actions A_G est partitionné en deux sous-ensembles :

- L'ensemble des actions continues A_C. Ces actions sont dites continues car la sortie associée à chacune d'elles n'est émise que lorsqu'une étape portant l'action est active.
- L'ensemble des actions mémorisées A_S. Ces actions sont dites mémorisées car le début ou l'arrêt de l'émission de la sortie associée à une telle action se fait en fonction d'ordres de type *Set* et *Reset* générés lorsque les étapes portant l'action sont actives.

Le modèle de spécification Grafcet permet de considérer des évolutions dites fugaces ([3], page 13) dont nous proposons une définition formelle ci-dessous :

Définition 1 : Evolution fugace
Soit un Grafcet G dans une situation (ensemble d'étapes actives) $S_{Act} \in S_G$.
Soit $T_{valid} \subset T_G$ l'ensemble de transitions validées, c'est à dire l'ensemble des transitions dont toutes les étapes amont sont actives $T_{valid} = \{t \in T_G | t.S_u \in S_{Act}\}$.
Si il existe une combinaison i_G de valeurs des variables d'entrée telle que :

- Il existe $t_1 \in T_{valid}$ telle que $t_1.FC(i_G, X_G) = True$
- Soit le nouvel ensemble d'étapes actives après le franchissement de la transition t_1, $S_{New} = (S_{Act}/t_1.S_u) \cup t_1.S_D$, il existe $t_2 \in T_G, t_2 \notin T_{valid}$ telle que $t_2.S_u \in S_{New}$ et $t_2.FC(i_G, X_G) = True$

Alors, l'évolution $t_1 t_2$ est dite fugace pour la valeur des variables d'entrée i_G et l'ensemble $t_1.S_D \cap t_2.S_u$ est l'ensemble des étapes instables.

Cette définition qui ne considère qu'une séquence de franchissement de deux transitions peut être étendue au cas d'une séquence de plus de deux transitions ou à des séquences comportant des franchissements simultanés de plusieurs transitions.

La figure 2 propose un exemple simple où la combinaison de valeurs des variables d'entrée entraîne ou non une évolution fugace du Grafcet.

De même, un Grafcet est dit dans une situation stable pour une combinaison de valeurs des variables d'entrée i_G si aucune transition n'est franchissable pour cette combinaison :

Définition 2 : Situation stable
Soit un Grafcet G comportant un ensemble d'étapes actives à l'instant considéré $S_{Act} \in S_G$.
Soit $T_{valid} \subset T_G$ l'ensemble de transitions validées $T_{valid} = \{t \in T_G | t.S_u \in S_{Act}\}$.
Soit i_G une combinaison de valeurs des variables d'entrée.
Si $\forall t \in T_{valid}$, $t.FC(i_G, X_G) = False$, alors le Grafcet est

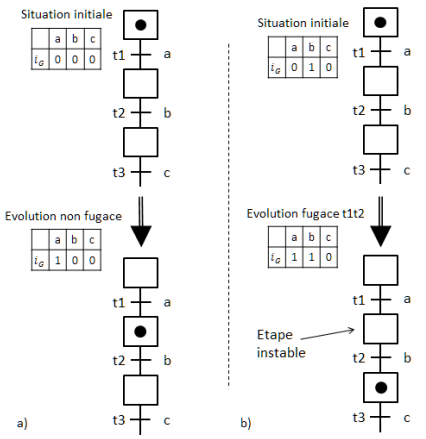

Fig. 2. Evolution non fugace (a) et fugace (b)

dit dans une situation stable (i.e. seul un changement de valeurs des variables d'entrée pourra déclencher le franchissement d'au moins une transition).

III. Execution du Grafcet par un contrôleur logique

A. Cycle de fonctionnement d'un contrôleur logique

Les contrôleurs logiques les plus couramment utilisés dans l'industrie pour implanter un Grafcet sont les Automates Programmable Industriels à scrutation cyclique ou périodique des entrées. Le mode de fonctionnement périodique (figure 3) se décompose en 4 phases :

- Lecture des entrées (L),
- Calcul des valeurs des variables internes et des sorties (C),
- Emission des sorties (E),
- Attente jusqu'à la fin de la période T.

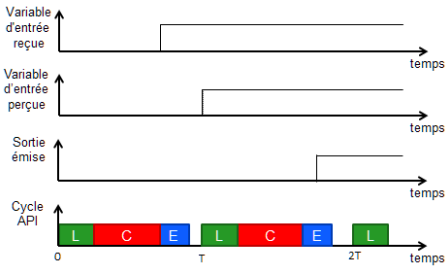

Fig. 3. Fonctionnement périodique d'un CLI

La phase d'attente est omise lorsque le fonctionnement est cyclique ou apériodique. Dans les deux cas, le cycle du contrôleur est contrôlé par un dispositif dit de chien de garde (watchdog) qui arrête l'exécution et positionne les sorties dans un état visant à assurer la sécurité lorsque la durée de la phase de calcul dépasse un seuil fixé.

B. Exécution avec recherche de stabilité

Selon la norme [3] définissant le Grafcet, le contrôleur doit exécuter la spécification Grafcet avec recherche de stabilité, c'est à dire que les sorties associées à des actions continues ne doivent être mises à jour que lorsque la situation du Grafcet pour la combinaison des valeurs des variables d'entrée courante est stable. Ce mode d'exécution est décrit pas l'algorithme 1.

while *Vrai* **do**
  Lire les entrées ;
  Stabilité := Faux ;
  while *Stabilité = Faux* **do**
    Déterminer l'ensemble des transitions franchissables ;
    Déterminer l'ensemble des nouvelles étapes actives ;
    Mettre à jour les actions mémorisées ;
    Déterminer l'ensemble des transitions franchissables ;
    if *l'ensemble des transitions franchissables est vide* **then**
      Stabilité := Vrai ;
    end
  end
  Mettre à jour les actions continues ;
  Emettre les sorties ;
end

Algorithm 1: Exécution du Grafcet avec recherche de stabilité

C. Exécution sans recherche de stabilité

En pratique, la recherche de stabilité génère une variation du temps de calcul des nouvelles valeurs des sorties qui peut dépasser le temps alloué à ces calculs dans un cycle du contrôleur. C'est pourquoi une exécution sans recherche de stabilité est généralement retenue pour les applications industrielles.

Ce mode d'exécution présenté Algorithme 2 possède, lui, un temps de calcul des sorties plus court car il ne comporte pas de boucle conditionnée par le caractère stable ou instable de la situation courante.

while *Vrai* **do**
  Lire les entrées ;
  Déterminer l'ensemble des transitions franchissables ;
  Déterminer l'ensemble des nouvelles étapes actives ;
  Mettre à jour les actions mémorisées ;
  Mettre à jour les actions continues ;
  Emettre les sorties ;
end

Algorithm 2: Exécution du Grafcet sans recherche de stabilité

IV. Construction du modèle formel du contrôleur

Cette section s'intéresse à la construction du modèle formel d'un contrôleur exécutant une spécification Grafcet sans ou avec recherche de stabilité. Ce modèle sera décrit sous la forme d'un automate des localités (AL), classe d'automate à états finis [5] dont chaque état, dénommé localité, regroupe les concepts de situation courante de la spécification, de sorties émises dans cette situation et de condition

de stabilité de la situation, et chaque transition entre deux localités, dénommée évolution, est étiquetée par une condition logique sur les variables d'entrée qui autorise cette évolution. La condition de stabilité incluse dans une localité est le complément de la somme logique des conditions associées aux évolutions qui partent de cette localité.

Pour prendre en compte les deux modes d'exécution de la spécification, deux algorithmes de construction seront proposés. On présentera dans un premier temps de manière détaillée un algorithme permettant de construire l'AL qui modélise un contrôleur exécutant une spécification sans recherche de stabilité ; le cas d'une exécution avec recherche de stabilité sera déduit de cet algorithme.

A. Définition formelle d'un AL

Un AL est défini comme un $5-uplet$
$AL = (I_{AL}, O_{AL}, L, l_{Init}, Evol)$ où :
- $I_{AL} = I_G$ est l'ensemble des entrées logiques,
- $O_{AL} = O_G$ est l'ensemble des sorties logiques,
- L est l'ensemble des localités,
- l_{Init} est la localité initiale,
- $Evol$ est l'ensemble des évolutions de l'AL.

Chaque localité $l \in L$ est définie par un $3-uplet$ $(S_{Act}, O_{Em}, E_{Stabl}(I_G))$ où :
- S_{Act} est la situation du Grafcet G,
- O_{Em} est un sous-ensemble des sorties de G définissant les sorties émises dans cette situation,
- $E_{Stabl}(I_G)$ est la condition de stabilité de la localité, sous la forme d'une expression booléenne sur I_G correspondant à la condition pour qu'aucune transition de G validée pour la situation courante ne soit franchissable.

Chaque évolution $evol \in Evol$ est définie par un $3-uplet$ $(l_U, l_D, E_{Evol}(I_G))$ où :
- $l_U \in L$ est la localité amont de l'évolution,
- $l_D \in L$ est la localité aval de l'évolution,
- $E_{Evol}(I_G)$ est la condition de franchissement de l'évolution, sous la forme d'une expression booléenne sur I_G correspondant à la condition sur I_G pour le franchissement d'un ensemble de transitions de G.

La Figure 4 illustre les informations portées par une localité et une évolution.

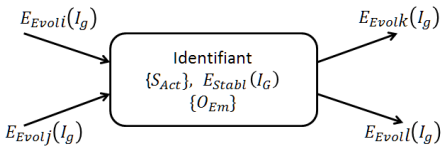

Fig. 4. Localité d'un AL

B. Construction d'un AL pour une exécution sans recherche de stabilité

Cette construction consiste globalement à déduire de la définition formelle de la spécification Grafcet et du mode d'exécution l'ensemble des éléments de l'AL. Elle peut être décrite par un algorithme décomposable en deux fonctions. La fonction principale (Algorithme 3) définit les ensembles des entrées et des sorties de l'AL, sa localité initiale, les sorties émises dans cette localité et appelle la fonction récursive NewLocations.

L'objectif de cette fonction (Algorithme 4) est de fournir la condition de stabilité d'une localité. Pour ce faire, l'ensemble des évolutions possibles depuis la localité courante est tout d'abord déterminé par analyse des transitions simultanément franchissables du Grafcet. La condition de stabilité de la localité, complément de la somme logique des conditions associées aux évolutions qui partent de cette localité, est alors aisément obtenue. La fonction est alors relancée pour les nouvelles localités atteintes suite à ces évolutions ; on construit ainsi l'AL pas à pas.

Données: Grafcet
Résultat: Location Automaton
$I_{AL} = I_G$;
$O_{AL} = O_G$;
$Oem_{Init} = EmitOut(Sinit)$;
$L_{Init} = CreateLocation(S_{Init}, Oem_{Init})$;
$L_{Init}.Estab = NewLocations(L_{Init})$;
Algorithm 3: Algorithme de génération d'AL

C. Construction d'un AL pour une exécution avec recherche de stabilité

Les deux fonctions présentées ci-dessus sont là encore utilisées mais la fonction CreateEvol, présente par deux fois dans la fonction NewLocations, mais non développée pour des raisons de place, doit alors rechercher les évolutions qui conduisent à des localités dont la condition de stabilité est satisfaite pour la valeur considérée des variables d'entrée, ce qui n'était pas le cas précédemment. L'AL généré ne contient que de telles localités et peut donc être nommé Automate des Localités Stables (ALS).

V. Exemple illustratif

Les résultats précédents sont illustrés dans cette section sur une étude de cas issue de [1]. Le système considéré est un système de remplissage/vidange de réservoirs (Figure 5) ; La commande de ce système comporte 5 entrées logiques (h_1, l_1, h_2, l_2, s) et 4 sorties logiques (V_1, W_1, V_2, W_2). Le fonctionnement démarre sur appui du bouton poussoir s qui déclenche le remplissage des deux réservoirs. Dès qu'ils sont pleins, les réservoirs se vident jusqu'au niveau bas. Une fois les deux réservoirs vidés, la commande se met en attente d'un nouvel appui sur le bouton-poussoir. Ce fonctionnement est spécifié par le Grafcet présenté Figure 6.

Les modèles de contrôleurs exécutant la spécification sans ou avec recherche de stabilité sont présentés Figure 7 et Figure 8. On peut remarquer deux grandes différences entre ces deux modèles :
- L'AL contient une localité de plus que l'ALS, qui correspond à la situation {4,7} du Grafcet toujours instable.
- L'ALS contient seize évolutions de plus que l'AL, ceci étant dû au fait que toutes les évolutions fugaces du Grafcet sont définies de manière explicite dans l'ALS alors qu'elles n'apparaissent pas dans l'AL.

```
Fonction : NewLocations
Entrée: L
Sortie: Estab
/* Détermine la condition de stabilité de la
   localité après avoir défini les évolutions
   possibles depuis celle-ci                    */
ActSteps = L.S_Act ;/* Etapes du Grafcet actives
   pour la localité considérée                  */
T_valid = ()/* Ensemble des transitions du Grafcet
   validées                                     */
for t ∈ T_G do Calcul de la condition de
franchissement de chacune des transitions
validées
    if t.S_U ∈ Actsteps then
        T_valid.add(t);
        t.E_franc = ∏_{s∈Actsteps} S.X · ∏_{s∉Actsteps} S.X̄ ·
            t.FC ;/* Calcul de la franchissabilité
            de la transition en fonction de la
            situation courante                  */
    end
end
SSFT = CreateSFT((ActSteps, T_valid))/* Ensemble
   des transitions simultanément franchissables
   */
for SFT ∈ SSFT do Détermination de la nouvelle
localité atteinte pour chaque évolution
    evol = 0;/* condition de franchissement de
       l'évolution                              */
    NewSteps = (ActSteps);
    new = True;/* Variable caractérisant les
       localités non traitées                   */
    for t ∈ T_valid do
        if t ∈ SFT then
            NewSteps.remove(t.s_U);
            NewSteps.add(t.s_D);
        end
    end
    evol =
       ∏_{t∈{Tvalid∩SFT}} t.FC · ∏_{t∈{Tvalid∩SFT̄}} t.FC̄;
       NewOem = EmitOut(NewSteps);/* Calcul
       des sorties émises                       */
    for l ∈ L do
        /* Détecte si la localité a déjà été
           rencontrée                           */
        if l.Sact = NewSteps and l.Oem = NewOem
        then
            new = False;
            CreateEvol(L, l, evol);
        end
    end
    /* Si non, définit la nouvelle localité et
       les évolutions possibles                 */
    if new = True then
        NewL = CreateLocation(NewSteps, NewOem);
        NewL.Estab = NewLocations(NewL);
        CreateEvol(L, NewL, evol);
    end
    Estab = Estab · evol̄;/* Met à jour la condition
       de stabilité de la localité              */
end
Retourner Estab
```

Algorithm 4: Fonction récursive de recherche de condition de stabilité

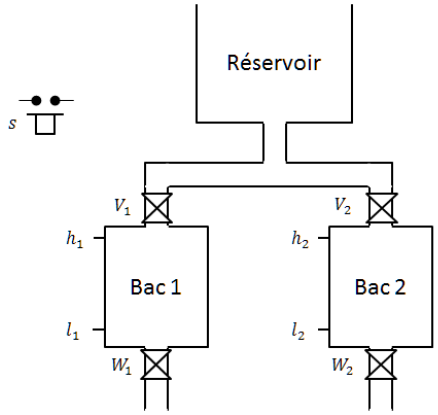

Fig. 5. Système de remplissage/vidange de réservoirs

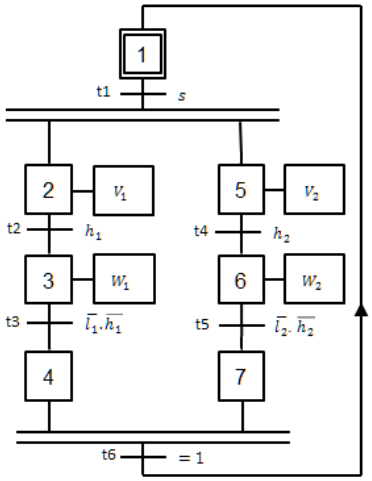

Fig. 6. Grafcet décrivant la commande du système de remplissage/vidange

VI. Utilisation d'un automate des localités pour le test de conformité

Un automate des localités peut être transformé en une machine de Mealy équivalente de laquelle il est possible de déduire une séquence de test couvrant au moins une fois chacune des transitions comme indiqué dans [6]. La séquence de test dépend donc de l'automate modélisant le contrôleur et il importe, pour obtenir des résultats de test cohérents, de connaître le mode d'exécution du contrôleur ou, en d'autres termes, de savoir, pour une spécification Grafcet donnée, s'il faut utiliser l'AL ou l'ALS comme modèle de départ pour la construction de la séquence.

Cette influence du modèle formel sur les résultats de test est illustrée dans ce qui suit à l'aide d'une séquence de trois pas donnée en Table 1. Cette séquence a été construite à partir de l'automate ALS de la figure 7, c.a.d. en supposant que le contrôleur exécute le Grafcet avec recherche de stabilité; elle correspond à des évolutions de la localité 0 à la localité 1, puis de la localité 1 à la localité 3, et enfin de la localité 3 à la localité 5. Dans cette table, la valeur 0 d'une variable signifie qu'elle est à l'état faux et la valeur 1 signifie qu'elle est à l'état vrai.

Chaque pas de test a une durée constante qui doit être supérieur au temps de cycle maximal du contrôleur, afin que chaque changement de vecteur d'entrée soit pris en

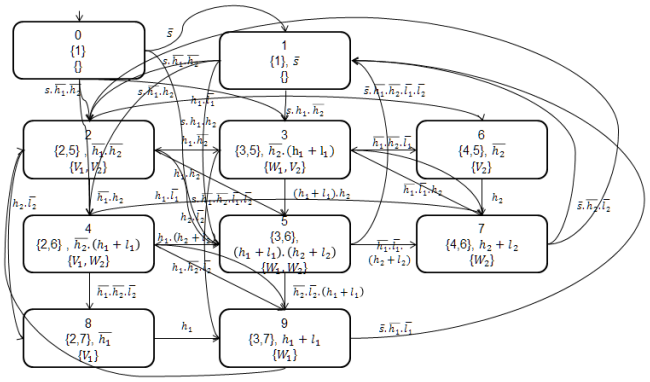

Fig. 7. ALS modélisant un CLI exécutant le Grafcet de la figure 6 avec recherche de stabilité

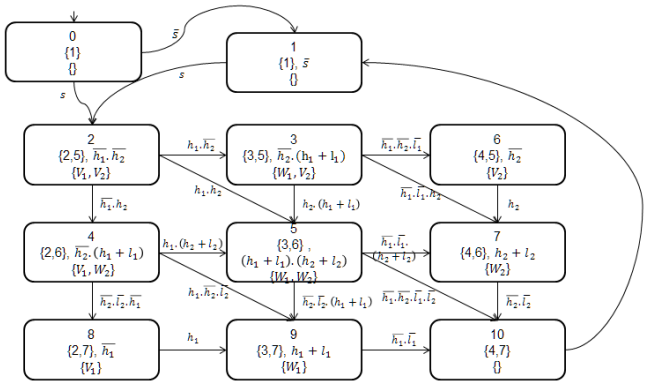

Fig. 8. AL modélisant un CLI exécutant le Grafcet de la figure 6 sans recherche de stabilité

compte.

Si cette séquence est utilisée pour le test d'un contrôleur qui exécute la même spécification mais sans recherche de stabilité (cas industriel le plus courant), les verdicts de test seront les suivants pour chacun des pas, en supposant que le contrôleur exécute correctement l'AL de la figure 8 :

- Pour le premier pas de test, la localité active devient la localité 1 et le vecteur de sortie observé O(1) = (0 ; 0 ; 0 ; 0) est identique au vecteur de sortie attendu ; le test est positif.
- Pour le second pas de test, la localité active devient tout d'abord la localité 2 et le vecteur de sortie observé O(2) = (1 ; 1 ; 0 ; 0) est différent de ce qui est attendu, ce qui était aisément prévisible car il n'existe pas d'évolution entre les localités 1 et 3 dans l'AL.

Deux cas sont alors possibles :

- si le changement de valeur des variables d'entrée (pas 3) se fait avant que l'AL ait pu évoluer de 2 à 3, la localité 4 est atteinte au pas 3 est le vecteur de sortie est O(3) = (1 ; 0 ; 0 ; 1),
- sinon, l'AL évolue de 2 à 3 avec pour vecteur de sortie O(3) = (0 ; 1 ; 1 ; 0) puis de 3 à 5 où le vecteur de sortie vaut O(4) = (0 ; 0 ; 1 ; 1).

Quel que soit le cas considéré, le test est négatif car la séquence de sortie observée diffère de celle attendue.

VII. Conclusions et perspectives

La construction du modèle formel d'un contrôleur logique nécessite de connaître non seulement la spécification

		Pas 1	Pas 2	Pas 3
Séquence d'entrée	s	0	1	0
	h_1	0	1	0
	h_2	0	0	1
	l_1	1	1	1
	l_2	1	1	1
Séquence de sortie attendue	V_1	0	0	0
	V_2	0	1	0
	W_1	0	1	1
	W_2	0	0	1
Séquence de sortie observée	V_1	0	1	1
	V_2	0	1	0
	W_1	0	0	0
	W_2	0	0	1
			contre	
			1 0	0
			1 1	0
			0 1	1
			0 0	1

TABLE I

Grafcet de la commande qu'il exécute mais encore le mode d'exécution de cette spécification. Ce papier a proposé une méthode de construction de ce modèle formel qui prend en compte ce mode. L'importance de cette connaissance pour la validité d'un test de conformité a ensuite été mise en évidence.

Nos travaux futurs s'orientent vers la validation de contrôleur à l'aide de techniques complémentaires au test de conformité, telles que l'identification d'un système bouclé contrôleur-processus ; le couplage du processus commandé au contrôleur permet en effet de limiter la taille de l'espace d'états et donc potentiellement la durée de la validation.

Références

[1] René David et Hassane Alla. *Du Grafcet aux réseaux de Pétri*. 2e éd. rev. augm. Traité des Nouvelles Technologies. France : Hermès, 2007.

[2] IEC 1012. *IEEE Standard for Software Verification and Validation*. Institute of Electrical et Electronics Engineers, 2004.

[3] IEC 60848. *GRAFCET Specification language for sequential function charts (2nd ed.)* International Electrotechnical Commission, 2002.

[4] D. Lee et M. Yannakakis. "Principles and methods of testing finite state machines-a survey". Dans : *Proceedings of the IEEE* 84.8 (août 1996), pp. 1090–1123. ISSN : 0018-9219.

[5] Julien Provost, Jean-Marc Roussel et Jean-Marc Faure. "A formal semantics for Grafcet specifications". Dans : *7th IEEE Conference on Automation Science and Engineering (IEEE CASE 2011)*. Trieste, Italie, août 2011, p. 488–494.

[6] Julien Provost, Jean-Marc Roussel et Jean-Marc Faure. "Translating Grafcet specifications into Mealy machines for conformance test purposes". Dans : *Control Engineering Practice* 19.9 (sept. 2011), pp. 947–957.