Synthèse d'une commande supervisée à base de contraintes booléennes

Abdelouahed Tajer, Pascale Marangé, François Gellot, Véronique Carré-Ménétrier

Synthèse d'une commande supervisée à base de contraintes booléennes

ABDELOUAHED TAJER, PASCALE MARANGE, FRANÇOIS GELLOT, VERONIQUE CARRE-MENETRIER

Centre de Recherche en STIC,
Université de Reims Champagne Ardenne, Moulin de la Housse, B.P.1039, 51687 REIMS Cedex2, FRANCE

{abdelouahed.tajer, pascale marange, francois.gellot, veronique.carre}@univ-reims.fr

***Résumé* — Une étape essentielle pour formaliser une méthode de synthèse de commande est la modélisation de la partie opérative (PO) et des contraintes. Cependant, cette étape reste une tâche très complexe. Pour pallier cette difficulté et dans le but d'aider le concepteur, nous proposons dans cet article une méthodologie de modélisation de la partie opérative sous forme de règles et une modélisation conviviale des contraintes par équations logiques dans l'algèbre de Boole classique. Pour la partie opérative, le résultat conduit à des modèles automates à états booléens et à des implications logiques pour les contraintes. Nous proposons ensuite un algorithme de synthèse en accord avec cette nouvelle modélisation. Un exemple de tri illustre ce papier.**

***Mots clés* — Automates booléens, modèle de partie opérative, modèle de contraintes, théorie de supervision, algèbre booléenne.**

I. INTRODUCTION

La complexité des Systèmes Automatisés de Production, accroît les exigences des utilisateurs en terme de sûreté de fonctionnement et d'aide à la conception de programmes. Pour aider le concepteur sur ces aspects, deux approches sont possibles : la première par validation et la seconde par synthèse. Nos travaux concernent l'approche par synthèse qui consiste à construire une commande où les propriétés attendues pour le système sont prises en compte dès le début de la conception. Ces travaux ont permis de caractériser les étapes nécessaires pour passer d'une spécification Grafcet [1] à l'implantation de la commande correspondante. Pour formaliser ces étapes, nous avons choisi d'asseoir notre démarche sur des outils/méthodes ayant de solides bases formelles, tels que les automates et la théorie de supervision des SED [2]. Ce contexte formel a été retenu en vue de démonter la faisabilité de l'approche en faisant dans un premier temps, abstraction des difficultés d'utilisation. Le travail a ainsi abouti au développement d'une approche globale de synthèse et d'implantation de la commande réactive la plus permissive vis-à-vis d'un Grafcet, d'un modèle du procédé à commander et des contraintes à respecter sur les évolutions du procédé. Nous avons montré l'applicabilité de cette approche sur plusieurs exemples dans [3] et [4]. Cependant, la modélisation de la partie opérative et des contraintes s'avère être une tâche complexe car la théorie de supervision préconise des modèles automates événementiels et rudimentaires. D'autre part, l'explosion combinatoire inhérente à ce type de modèles contraint l'utilisation de notre démarche de synthèse à des systèmes simples en terme du nombre d'entrées/sorties.

Pour pallier ces problèmes, nous proposons d'intégrer dans la démarche des modèles structurés et évolués en utilisant pour la partie opérative, des modèles booléens à base de règles et pour les contraintes, des équations logiques dans l'algèbre de Boole classique. Une aide dans l'élaboration de la commande et dans le raffinement des modèles de départ est proposée au concepteur pour lui permettre de visualiser et d'analyser les séquences bloquantes ainsi que les corrections proposées [5]. La méthodologie employée pour la partie opérative peut être considérée comme une adaptation de l'approche de V. Chandra [6], c'est la structuration qui est ici recherchée. Pour les contraintes, l'utilisation d'une modélisation par des équations logiques dans l'algèbre de Boole classique entre les entrées/sorties du contrôleur est adoptée. C'est la convivialité du modèle et le fait de réduire l'explosion combinatoire qui est ici recherché. Le schéma de la figure 1 tient compte de ces modifications et présente la démarche dans son ensemble.

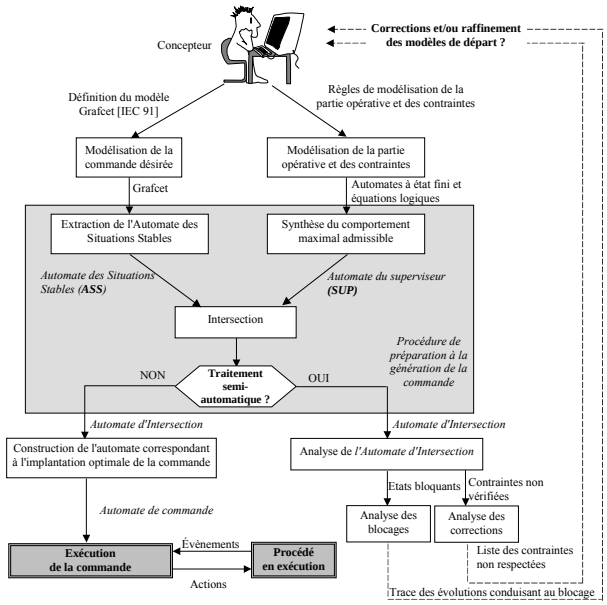


Fig. 1 : Démarche de synthèse formelle à partir de spécifications de la commande exprimée en Grafcet

Après la phase de modélisation, la commande spécifiée sous forme de Grafcet est convertie en Automate des Situations Stables (ASS). Pour tenir compte des modèles de partie opérative et de contraintes, un algorithme de synthèse est proposé selon les principes énoncés par la théorie de supervision, générant ainsi le superviseur (SUP) représentant le comportement maximal admissible du procédé. Ensuite, une opération d'intersection permet d'obtenir un automate correspondant au comportement commun entre l'automate de des situations stables et l'automate du superviseur. Pour intégrer le concepteur dans la boucle d'élaboration de la commande et ainsi rendre la démarche moins sensible aux erreurs de modélisation, il est proposé d'effectuer la construction soit de manière automatique, soit de manière semi-automatique. Dans ce mode, le concepteur peut visualiser par exemple, la trace des évolutions de la commande conduisant à un blocage. Ces informations lui permettent alors d'agir en relâchant des contraintes peut-être trop restrictives, en affinant le modèle de la partie opérative ou en modifiant la commande proprement dite. De nouvelles itérations de la

synthèse sont alors effectuées à partir des modèles corrigés pour générer en finalité un modèle de commande correcte implantable.

En section 2, nous présentons la méthodologie de modélisation de la partie opérative et en section 3, la modélisation des contraintes par équations booléennes. La section 4 détaille l'algorithme de synthèse du superviseur et en section 5, nous proposons d'utiliser la démarche dans le cadre de la validation de commande.

Nous avons choisi d'illustrer ces travaux à travers une application dont l'objectif est de trier automatiquement des caisses de deux tailles différentes (Figure 2). Le système se compose du convoyeur 1 qui apporte les caisses les unes après les autres, d'un vérin composé de deux vérins double effet V1 et V2, de deux vérins simple effet V3 et V4, et de deux convoyeurs (convoyeur 2 et 3) permettant l'évacuation des caisses. Selon la taille de la caisse, le vérin composé de V1 et de V2, place les caisses devant le vérin V3 ou devant le vérin V4.

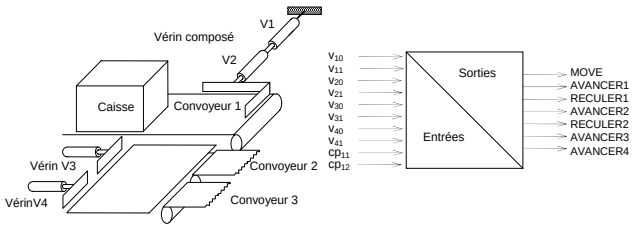


Fig. 2 : Système du tri de caisses

Dans la suite du papier, nous utilisons les variables suivantes : v_{i0} pour la position d'entrée du vérin v_i, v_{i1} pour la position de sortie du vérin v_i, cp_{11} pour le capteur de la petite caisse et cp_{12} pour le capteur de la grande caisse. Les sorties du système sont définies par : MOVE correspond à l'activation des 3 convoyeurs, *AVANCER1 (2)* : ordre de sortie du vérin V1 (V2), *RECULER1 (2)* : ordre de rentrée du vérin V1 (V2), *AVANCER3 (4)* : ordre de sortie du vérin simple effet V3 (V4). Les vecteurs d'entrées et de sorties correspondant au système de tri de caisses sont définis de la façon suivante :

E = [v_{10}, v_{11}, v_{20}, v_{21}, v_{30}, v_{31}, v_{40}, v_{41}, cp_{11}, cp_{12}],
Z = [MOVE, AVANCER1, AVANCER2, RECULER1, RECULER2, AVANCER3, AVANCER4].

II. Modelisation de la Partie Operative

Nos travaux précédents ont montré la nécessité d'un modèle du comportement de la partie opérative pour identifier et traiter les blocages de la commande, conduisant ainsi à exprimer une image explicite de chaque élément de la partie opérative. Cependant, la description précise du comportement de la partie opérative est une opération complexe car les évolutions d'un système physique sont de nature asynchrone et non déterministe. Pour contourner les difficultés d'une modélisation globale, nous avons choisi une démarche modulaire permettant d'exprimer des causalités simples entre les éléments de la partie opérative en fonctionnement normal. Par conséquent, la modélisation s'effectue à l'aide d'un automate réceptif aux ordres de la commande et qui agit sur l'évolution des valeurs logiques des entrées du Grafcet de manière à exprimer les réactions de la partie opérative.

Au vu de la complexité des systèmes, une méthodologie est nécessaire pour obtenir le modèle de la partie opérative. Cette méthodologie est basée sur la définition de règles d'occurrence et de relations de précédence entre les différents éléments de la partie opérative. Ces règles définissent les interactions entre les événements commandables ou non commandables. Les relations de précédence indiquent les liens entre les événements non commandables. L'utilisateur définit les règles et les relations qui sont ensuite traduites en automates compatibles avec notre approche de synthèse.

A. Cadre formel

Pour conserver le bénéfice du cadre formel proposé par la théorie de supervision [2], la modélisation de la partie opérative s'effectue sous forme d'automates décrivant les évolutions physiquement possibles en fonctionnement normal du procédé par des événements simples. Pour refléter les interactions entre la commande spécifiée par Grafcet et la partie opérative, nous avons choisi l'interprétation de S. Balemi [7], où les événements commandables Σ_c représentent les entrées du procédé et les événements non commandables Σ_u ses sorties. La commande peut dès lors forcer à tout instant l'entrée du procédé et la génération des événements est initiée conjointement par le procédé et/ou la commande. Cette interprétation est spécialisée de manière à concilier la nature continue des actions et des variables d'entrées du Grafcet avec le caractère événementiel du modèle de la théorie. La correspondance suivante a été retenue : un événement commandable correspond soit à l'activation $\uparrow z$ soit à la désactivation $\downarrow z$ d'un ordre du Grafcet, tandis qu'un événement non commandable est associé soit au front montant $\uparrow e$ soit au front descendant $\downarrow e$ d'une variable d'entrée du Grafcet [4].

B. Règles d'occurrence et de précédence

Pour déterminer les règles d'occurrence [4], il est nécessaire (i) de fixer les conditions initiales du système, (ii) de déterminer tous les événements liés à l'élément de la partie opérative, (iii) de définir avec les règles, l'influence des événements commandables sur les événements non commandables. Chaque règle exprime une "cause/conséquence" ; la cause relie l'événement commandable à la conséquence qui est un événement non commandable. Pour chaque règle d'occurrence ayant la même cause, une relation de précédence est établie et concerne la chronologie entre les événements non commandables conséquents. Ces règles d'occurrence et ces relations de précédence obtenues vont permettre par la suite de déterminer, à partir de l'état initial, les évolutions commandables et non commandables dans l'automate de l'élément de la partie opérative à modéliser.

Condition initiale	Règles d'occurrence	Relations de précédence
$\overline{AVANCER2}$	$\uparrow AVANCER2 \rightarrow \downarrow v_{20}$ $\uparrow AVANCER2 \rightarrow \uparrow v_{21}$	$\downarrow v_{20}$ *précède* $\uparrow v_{21}$
$\overline{RECULER2}$	$\uparrow RECULER2 \rightarrow \downarrow v_{21}$ $\uparrow RECULER2 \rightarrow \uparrow v_{20}$	$\downarrow v_{21}$ *précède* $\uparrow v_{20}$
$\overline{v_{21}}$	$\downarrow AVANCER2 \rightarrow \uparrow v_{21}$	*Aucune*
v_{20}	$\downarrow RECULER2$ $\uparrow v_{20}$	*Aucune*

Tableau 1. Paramètre du mouvement du vérin V2

Pour l'exemple, le modèle complet de partie opérative se compose de 4 automates décrivant respectivement le mouvement de chaque vérin, ces modèles tenant compte de la technologie du vérin. Pour le mouvement du vérin double effet V2, il y a quatre événements commandables $\downarrow AVANCER2$, $\uparrow AVANCER2$, $\downarrow RECULER2$, et $\uparrow RECULER2$ et quatre événements non commandables $\uparrow v_{20}$, $\downarrow v_{20}$, $\uparrow v_{21}$ et $\downarrow v_{21}$. Dans la situation initiale, le vérin est en position rentrée et aucun ordre n'est envoyé à la partie opérative. Pour définir les règles d'occurrence, il est nécessaire d'observer les conséquences des ordres envoyés. L'ordre *AVANCER2* rend possible le déplacement du vérin, la chronologie est la suivante :

désactivation de v_{20} puis de l'activation de v_{21}. Les paramètres sont présentés dans le tableau 1.

C. Construction de l'automate de partie opérative

La structure des automates à états rudimentaires adoptée pour les SED et introduite par la théorie de supervision [2] n'est pas très explicite car les états ne portent pas toutes les informations sur le système. Il est intéressant d'enrichir le modèle en terme de vecteurs d'entrées et de sorties associés à chaque état de l'automate. En effet, les vecteurs d'entrée et de sortie donnent une information sur l'état de la partie opérative et sur l'état de la commande. C'est pour cette raison que nous proposons de définir un automate à état booléen [8].

Un automate à état booléen est défini par un 4-uplet $G = (P, \sum, \delta, p_0)$, où $\sum = \sum_u \cup \sum_c$ est l'ensemble des événements décrivant les transitions. Ces transitions sont caractérisées par des événements commandables $\sum_c$ ou non commandables $\sum_u$, δ la fonction de transition définie par $\delta : P \times \sum \rightarrow P$, l'état initial p_0 de l'automate G et enfin P l'ensemble des états de G défini par :

$$P = \{[p_1,\ldots, p_n] / p_j \in \{0,1\}, j = 1,\ldots, n\} \subseteq IB^n$$

P est un ensemble de vecteurs d'états booléens à n dimensions. Chaque composante du vecteur d'état peut prendre la valeur 0 ou 1. Le nombre d'états maximum du modèle automate est donné par le nombre de variables d'états, c'est-à-dire 2^n états au maximum pour n variables.

A chaque état p de l'automate G est associé le vecteur d'entrées $E = [e_1,..., e_m]$, le vecteur de sorties $Z = [z_1..., z_l]$ et le label x_k où m est le nombre d'entrées du système, l le nombre de sorties du système et k l'indice de l'état de l'automate G.

L'ensemble des états de G est défini par :

$P = \{[[z_1,\ldots, z_n], [e_1,\ldots, em], x_k] / z_j, e_i \in \{0,1\}*\{0,1\}, j=1,...,l ;$
$i=1,\ldots,m$ et $k=1,...,n\} \subseteq IB^n$, n : le nombre d'états dans G.

La correspondance suivante est recommandée : l'activation d'un événement commandable correspond au passage à 1 (set) du composant correspondant dans le vecteur Z, la désactivation d'un événement commandable correspond au passage à 0 (reset). De la même façon, les événements non commandables sont traités pour la mise à jour du vecteur d'entrées E.

D. Méthode de construction proposée

La construction automatique de l'automate de la partie opérative est basée sur les règles et les relations et s'effectue selon les 4 étapes suivantes :

1. Construction de la table avec les 2^n états (n est le nombre d'entrées/sorties du système) décrivant toutes les combinaisons possibles entre les entrées/sorties du système
2. Suppression des incohérences logiques entre les entrées. Ici, nous ne nous intéressons pas aux incohérences logiques liées aux événements commandables mais à celles liées aux conséquences (événements non commandables) car seules les entrées de la partie opérative sont considérées.
3. Construction de l'automate équivalent à partir du diagramme des évolutions commandables. Ce diagramme est déterminé à partir de la table de vérité représentant les états restant après la suppression des incohérences logiques. Pour chaque combinaison, l'occurrence d'un événement commandable permet de changer la variable d'état de sortie ce qui conduit à un nouvel état.
4. Complément de l'automate avec les évolutions non commandables. Cette opération consiste à utiliser les règles d'occurrence et les relations de précédence ainsi que les conditions initiales.

Pour le mouvement du vérin V2, la première étape consiste à établir les 2^4 combinaisons entre AVANCER2, RECULER2, v_{21} et v_{20} (figure 3.a). La deuxième étape consiste à supprimer les 4 incohérences logiques d'événements liées au fait que v_{20} et v_{21} ne peuvent pas être vrais en même temps. L'automate des évolutions commandables est ensuite déterminé à partir du diagramme des évolutions commandables exprimant de ce fait toutes les évolutions possibles entre les états (figure 3.b). Ces évolutions sont définies uniquement par les événements commandables : *↑AVANCER2, ↓AVANCER2, ↑RECULER2, ↓RECULER2*. Sachant que *AVANCER2, RECULER2*, peuvent soit prendre la valeur 1 pour l'activation, soit 0 pour la désactivation (figure 3.c). Les relations de précédence engendrent dans la dernière étape, l'automate final présenté figure 3.d. A partir de l'état initial (2) un événement commandable possible sortant est *↑AVANCER2*. Son occurrence conduit à l'état 10. D'après le tableau 1, la conséquence de *↑AVANCER2* est l'apparition de l'événement non commandable $\downarrow v_{20}$ ce qui permet d'atteindre l'état 8. Les autres évolutions non commandables sont ajoutées à l'automate de base de manière identique.

Cette méthode de modélisation est appliquée à chacune des parties du système. Le modèle complet de la partie opérative est obtenu alors par la composition asynchrone de tous les modèles élémentaires.

Pour l'exemple, l'automate de la partie opérative complet obtenu par le produit asynchrone des modèles élémentaires se compose alors de 15552 états et de 163295 transitions.

états	*AVANCER2*	*RECULER2*	v_{20}	v_{21}	
0	0	0	0	0	
1	0	0	0	1	
2*	0	0	1	0	
3	0	0	1	1	⇒ *Incohérence logique*
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	⇒ *Incohérence logique*
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	
11	1	0	1	1	⇒ *Incohérence logique*
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	⇒ *Incohérence logique*

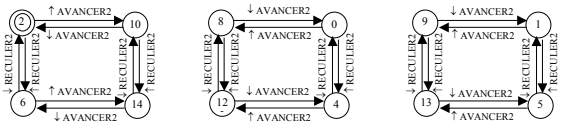
(*) État initial

-a- Table de vérité

AV2/RE2 \ v_{20}/v_{21}	00	01	11	10
00	0	1		❷
01	4	5		6
11	12	13		14
10	8	9		10

-b- Diagramme des évolutions

-c- Automate sans les évolutions non commandables



-d- Automate complet

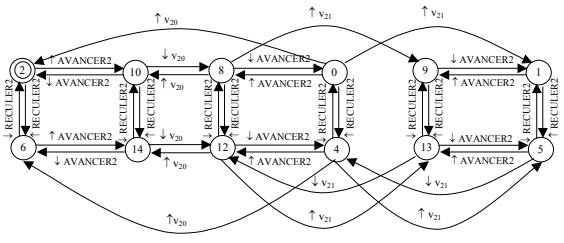


Fig 3: Automate du mouvement du vérin V2

III. MODELISATION DES CONTRAINTES

Les contraintes à modéliser sont de deux types : des contraintes de sécurité (ce qu'il ne faut pas faire) et des contraintes de vivacité (ce qu'il faut faire). Intégrer des contraintes consiste à inhiber des actions et/ou à agencer et séquencer l'exécution des commandes envoyées au procédé. Nous préconisons pour la modélisation de ces contraintes, l'utilisation d'équations logiques qui sont fonctions de l'ensemble des entrées et de l'ensemble des sorties du Grafcet [9]. L'utilisation des équations logiques à la place de l'automate réduit le risque d'explosion combinatoire. Des travaux utilisent également ce cadre formel [10] en considérant une approche algébrique basée sur les signaux binaires et prenant en compte des propriétés liées à l'aspect temporel dont nous faisons abstraction.

L'emploi des équations logiques booléennes a pour avantage d'exprimer les contraintes de manière explicite et flexible, dans un langage bien connu du concepteur. L'écriture de ces contraintes se détermine simplement en utilisant le connecteur d'implication « *Si …Alors…* » entre les événements.

Par exemple, pour le mouvement V2, le vérin ne peut pas rentrer et sortir en même temps, c'est-à-dire que l'envoi simultané des deux ordres *AVANCER2* et *RECULER2* est interdit. Si nous utilisons une modélisation sous forme d'automate pour représenter cette spécification, le modèle s'exprime par trois états (figure 4.a). Il est à comparer à l'équation logique de la figure 4.b. obtenue à travers l'implication suivante : « *Si AVANCER2* = 1 *Alors RECULER2* = 0 » ou « *Si RECULER2* = 1 *Alors AVANCER2* = 0 ». Ceci peut aussi s'écrire par une expression logique qui est le complément du ET logique entre *AVANCER2* et *RECULER2*.

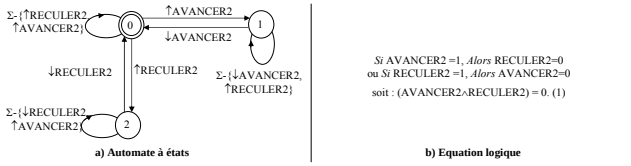


Fig. 4. Ne pas *AVANCER2* et *RECULER2* en même temps

Pour l'exemple du tri de caisses, deux autres contraintes ont été définies. La première contrainte interdit de sortir et de rentrer le vérin V1 simultanément : $AVANCER1 \wedge RECULER1 = 0$ (2), la deuxième contrainte fait référence à l'interdiction de sortir le vérin V3 si le vérin V1 est déjà sorti : $AVANCER3 \wedge v_{11} = 0$ (3)

IV. SYNTHESE DU SUPERVISEUR

A. Principe

Le principe de commandabilité définie par Ramadge et Wonham s'applique aux SED booléens. Il s'agit pour un procédé G défini par le 4-tuplet (P, Σ, δ, p_0), de vérifier la proposition suivante [8] :

$\forall p \in P$, tel que p vérifie K
Si $\exists \sigma \in \Sigma_u$ tel que $\delta(p,\sigma)$ vérifie K
Alors K est commandable, K est l'ensemble des spécifications à respecter par le système.

En d'autres termes, la spécification K est commandable si et seulement si l'occurrence d'un événement non commandable σ à partir d'un état vérifiant K ne conduit pas le procédé hors de la spécification K, c'est-à-dire vers un état qui ne vérifie pas K. Pour faciliter la mise en pratique de cette proposition, nous avons choisi, pour l'algorithme de synthèse [5], d'utiliser la notion d'états défendus et faiblement défendus où les états défendus représentent des états ne respectant pas la spécification K lors de l'occurrence d'un événement non commandable σ et les états faiblement défendus représentent des états conduisant vers un état défendu sous l'occurrence d'un événement non commandable σ.

B. Algorithme de synthèse

L'opération de synthèse proposée consiste, à partir du modèle global du procédé et des contraintes modélisées selon les principes précédents, à générer l'automate à état booléen du superviseur *SUP*. L'automate *SUP* est décrit par 6-uplet (Σ, Q, Δ, E, Z, q_0), avec Σ l'ensemble des événements, Q l'ensemble des états, q_0 l'état initial et Δ une fonction de transition partielle $\Delta: Q \times \Sigma \rightarrow Q$, un vecteur d'entrées E, et un vecteur de sorties Z. Une transition de l'automate *SUP* est définie par un triplet (q, σ, q'), q étant l'état de départ, σ l'événement correspondant à la transition et q' l'état d'arrivée. La fonction de transition peut être étendue aux séquences d'événements de la manière suivante : Soit Σ* l'ensemble des séquences d'événements de longueur finie, ε la séquence vide et w une séquence finie d'événements :

$\Delta : Q \times \Sigma^* \rightarrow Q$, $\Delta(q, \varepsilon) = q$ et $\Delta(q, w\sigma) = \Delta(\Delta(q,w), \sigma)$

Cet algorithme reçoit en entrée un automate à état booléen G représentant le modèle global du procédé, un ensemble K de spécifications logiques représentant les contraintes de sûreté et de vivacité ainsi que la structure initiale de l'automate résultant donnée par (Σ, $\{q_0\}$, δ, q_0). Un état q de *SUP* est caractérisé par (état de G, état du vecteur E, état du vecteur Z) enfin $q_0 = (p_0, E_0, Z_0)$.

a- Premier pas :

$\forall q \in Q$ faire :

Si $\exists s \in \Sigma_c$ tel que $\Delta(q,s) = q'$ **Alors**
  Si $\mathcal{K}_s$ est vrai **Alors**
  - $\Delta = \Delta \cup \{(q, s, q')\}$;
    $Q = Q \cup \{q'\}$ Si $q' \notin Q$.
  Sinon
    (q, s, q') est une évolution défendue

b- Deuxième pas :

1- **Si** $\exists \Delta(q,\sigma) = q'$ ($q' \in Q$, $\sigma \in \Sigma_u$) tel que q' ne vérifie pas $\mathcal{K}$
  Alors q est défendu

2- **Si** $\neg\exists q \in Q$ tel que q un état défendu
  Alors Fin.
  Sinon $\forall q \in Q$
    Si
      - q n'est pas défendu,
      - $\exists w \in \Sigma^*$ tel que ($\forall \sigma \in w, \sigma \in \Sigma_u$) et ($q_w = \Delta(q, w)$ et q_w est défendu)
    Alors q est faiblement défendu.

c- Troisième pas :

$\forall q \in Q$ tel que q est un état défendu ou faiblement défendu.
  - $\forall (q, \sigma, q') \in \Delta$, $\Delta = \Delta - \{(q', \sigma, q)\}$.
  - $\forall (q', \sigma, q) \in \Delta$, $\Delta = \Delta - \{(q, \sigma, q')\}$.
  - $Q = Q - \{q\}$.
  - $\forall q' \in Q$ tel que $\neg\exists(w \in \Sigma^*$ et $q' = \Delta(q_0, w))$ faire $Q = Q - \{q'\}$

Fig.5. Algorithme de synthèse

Cet algorithme est basé sur trois étapes :

La première étape (figure 5.a) traite les événements commandables liés à l'activation ou à la désactivation d'une sortie de la commande (↓↑z). Ces événements décrivent les ordres envoyés de la commande vers la partie opérative. Les contraintes sont structurées en fonction des événements commandables Σ_c. Ainsi, pour chaque événement commandable $s \in \Sigma_c$ est associé un ensemble de contraintes K_s, $K_s \subset K$. Par conséquent, une itération consiste, pour une évolution commandable ((q, s, q'), ($s \in \Sigma_c$)), à vérifier l'ensemble des contraintes K_s. Cette vérification est obtenue par la valeur du vecteur d'entrées E_q, du vecteur de sorties Z_q et de l'événement *s* (l'occurrence de l'événement commandable *s*

permet de changer la valeur de l'élément correspondant dans le vecteur Z pour obtenir $Z_{q'}$).

– Si dans l'ensemble K_s, les équations logiques associées à *s* sont vraies alors l'évolution correspondante n'est pas défendue et la transition correspondante est ajoutée à l'ensemble des transitions Δ.

– Si l'ensemble des contraintes K_s n'est pas vérifié (les équations ne sont pas vraies), alors l'évolution (q, s, q') est défendue.

La deuxième étape (figure 5.b) consiste à construire s'il n'existe pas déjà, à partir d'un état courant q, les évolutions de *SUP* caractérisées par des événements non commandables. Ces événements non commandables correspondent aux réactions de la partie opérative par rapport aux ordres de la commande. Dans cette étape, il y a deux pas :

1. Le premier pas concerne l'identification des états défendus conduisant vers des états à interdire où les contraintes K ne sont pas vraies. Pour cet état q, il existe un événement non commandable σ possible à partir de l'état correspondant du procédé q' tel que l'état atteignable ne vérifie pas K.
2. Le deuxième pas sert à identifier les états faiblement défendus. Ce sont tous les états à partir desquels il existe une séquence d'événements *w* non commandables permettant d'atteindre un état défendu.

Enfin, la dernière étape (figure 5.c) consiste à retirer dans un premier temps tous les états défendus et faiblement défendus ainsi que toutes les transitions amont et aval qui leurs sont associées. Dans un second temps, les états non atteignables à partir de l'état initial sont retirés de l'automate obtenu.

V. AIDE A L'ELABORATION DE LA COMMANDE

Jusqu'à présent, les travaux réalisés ont eu pour objectif l'implantation de la commande en suivant une démarche de synthèse automatique, ce qui signifie qu'aucune information n'était communiquée au concepteur sur les corrections effectuées avant implantation. Pour une réelle intégration dans la boucle d'élaboration de la commande (démarche dite semi-automatique, figure 1), nous proposons au concepteur de visualiser les séquences bloquantes et les corrections proposées pour lui permettre d'agir en modifiant par exemple la commande, en relâchant des contraintes ou en affinant le modèle de la partie opérative [5]. Pour conserver une démarche sécuritaire et générer en finalité un modèle de commande correcte, de nouvelles itérations de la synthèse doivent ensuite être effectuées à partir des modèles corrigés. Les informations sur les blocages en exécution réelle et les corrections (c'est à dire les inhibitions d'actions) apportées à la commande par la prise en compte des contraintes sont visibles au niveau de l'opération d'intersection entre *ASS* et *SUP*. Ainsi, à l'issue de l'opération d'intersection, dans la démarche dite semi-automatique, deux cas peuvent alors se produire :

– Aucune correction n'a été apportée par la démarche ce qui signifie que la commande respecte les spécifications imposées au système. Le concepteur peut implanter la commande synthétisée dans un automate programmable industriel (A.P.I.) par exemple, en toute sécurité. Il est ainsi certain qu'il n'y a aucun risque pour le matériel opératif, les opérateurs ou les produits.

– Des corrections ou des blocages sont détectés, la commande ne respecte donc pas les spécifications imposées au système :

* S'il s'agit d'une situation de blocage, le concepteur analyse alors la situation bloquante grâce aux données élaborées à partir des informations contenues dans les régions de l'automate d'intersection et agit en conséquence.

* S'il s'agit de corrections proposées sous forme d'inhibitions d'ordres ceci signifie que les contraintes imposées au système ne sont pas respectées. Le concepteur visualise alors une liste des contraintes non respectées au niveau de sa commande originelle et en déduit d'éventuelles modifications. Selon l'interprétation, il peut s'agir de contraintes trop restrictives ou d'un modèle de commande erroné fourni par le concepteur.

A. *Démarche d'analyse*

Pour structurer la démarche d'analyse, nous proposons au concepteur de procéder étape par étape en commençant par la spécification de la commande Grafcet jusqu'au modèle de bas niveau de la partie opérative. Les différentes étapes proposées sont les suivantes :

– La 1ère étape consiste à analyser la spécification Grafcet proposée en connaissant la situation complète du système grâce aux informations sur la valeur du vecteur d'entrées et de sorties, ainsi que les étapes actives dans le Grafcet. Cette analyse permet de comprendre les différentes évolutions possibles au niveau de la commande, c'est-à-dire, les différentes transitions franchissables.

– Si la 1ère étape ne permet pas au concepteur de détecter l'origine du blocage, il est informé sur les ordres autorisés et interdits qui déterminent si les contraintes sont respectées ou non dans l'état concerné. Il peut être nécessaire dans cette étape d'afficher le superviseur *SUP*.

– La 3ème information permettant de renforcer l'analyse du concepteur concerne les sous-modules de partie opérative. Les éléments (capteurs et actionneurs) mis en cause dans un début d'analyse pourront être visualisés dans le ou les sous-modules auxquels ils appartiennent.

B. *Etude de cas*

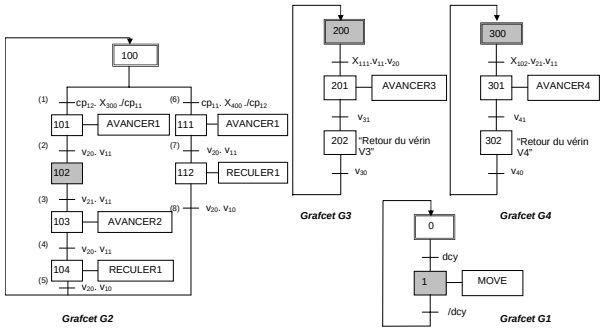


Fig. 6. Grafcet de commande de l'exemple de tri de caisses

Dans cette partie, nous analysons une erreur fréquemment recensée qui est l'oubli d'envoi d'un ordre dans la commande. La commande proposée par le concepteur, est celle de la figure 6 qui conduit vers un blocage comme le montre la figure 7. En effet, depuis l'état 8, aucune évolution n'est possible. La trace conduisant au blocage est proposée au concepteur.

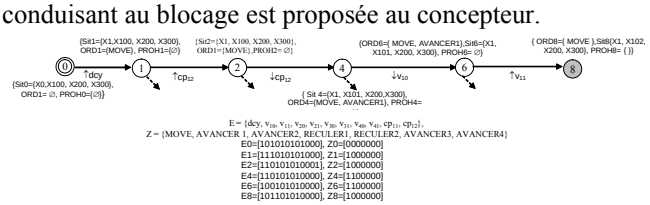


Fig. 7. Visualisation de la séquence de blocage

Dans l'état 8, la commande se trouve dans la situation {X1, X102, X200, X300} du Grafcet. Quant à la partie opérative, le vecteur d'entrées E_8= [1,0,1,1,0,1,0,1,0,0,0,0] et le vecteur de sorties Z_8= [1,0,0,0,0,0,0] indiquent que le système est en marche (*dcy = 1*), le vérin V1 est en position sortie, et les vérins V2, V3, et V4 sont en position rentrée. Au niveau des sorties, seul l'ordre *MOVE* est envoyé.

1ère étape : Analyse du Grafcet de commande

Par l'analyse du Grafcet, le concepteur peut en déduire que les évolutions possibles au niveau de la commande sont soit le passage à 0 de *dcy*, soit la validation et le franchissement de la transition notée (3) dans le Grafcet G2 dont la réceptivité est $v_{21}.v_{11}$. Pour que la réceptivité $v_{21}.v_{11}$ soit vraie, le système se trouvant dans la configuration où le vérin V2 est rentré, il faut que l'ordre *AVANCER2* soit envoyé pour faire sortir le vérin V2 et donc, par conséquent, activer la position v_{21}. A ce stade, soit le concepteur détermine directement l'erreur (il manque l'ordre *AVANCER2* dans l'étape 102) soit, il a besoin d'information supplémentaire lui permettant de vérifier que cet ordre n'est pas interdit par les contraintes.

2ème étape : Analyse du superviseur

Si le concepteur le désire, les informations provenant du superviseur, peuvent lui être fournies. L'état du superviseur correspondant à l'état 8 (figure 7), est l'état q_{16} (figure 8) où les ordres autorisés sont : *AVANCER2*, …

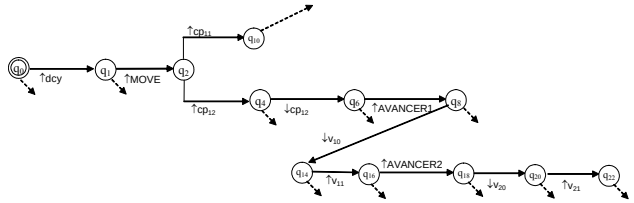


Fig. 8. Extrait de l'automate superviseur

L'ordre *AVANCER2* n'est donc pas interdit au niveau de l'automate de la partie opérative contrainte. Donc, soit le concepteur détermine que l'erreur se situe au niveau de la commande soit, il a besoin d'une information plus précise sur l'élément de la partie opérative associée à l'événement non commandable v_{21}.

3ème étape : Analyse de la partie opérative.

Pour être certain de son raisonnement, le concepteur peut demander à visualiser l'élément de partie opérative qui l'intéresse. A partir de cette dernière information (figure 3.d), le concepteur a confirmation que pour avoir l'occurrence de v_{21}, il faut l'envoi de *AVANCER2*. Le concepteur peut donc en conclure que pour que la réceptivité $v_{21}.v_{11}$ soit vraie, il faut que l'ordre *AVANCER2* soit envoyé vers la partie opérative, qu'aucune contrainte ne contredit à ce stade l'envoi de l'ordre et que seul un oubli de l'envoi de l'ordre au niveau de la commande génère le blocage.

VI. Conclusion

La mise en pratique des concepts de la théorie de supervision sur des exemples diversifiés, nous a permis de soulever plusieurs problèmes. Ce papier tente d'apporter une réponse au manque d'expressivité et de convivialité des modèles à états préconisés dans cette théorie, aux difficultés de modélisation et aux risques d'explosion combinatoire inhérents à leur utilisation. En effet, nous proposons une méthodologie modulaire à base de règles pour concevoir le modèle de la partie opérative et pour l'écriture des contraintes, une modélisation sous forme d'équations logiques. Nous avons proposé un algorithme de synthèse prenant en charge ces nouveaux modèles pour obtenir l'automate du superviseur. Cependant, l'utilisation d'équations logiques pour exprimer les contraintes ne nous permet pas actuellement d'exprimer des contraintes de types séquence d'événements ou temporelles. Nos travaux actuels se situent au niveau des modèles de partie opérative en considérant la technologie des pré-actionneurs, des actionneurs et des capteurs [11].

L'autre point que nous avons développé dans ce papier, est la possibilité d'utiliser l'approche de synthèse pour procurer au concepteur, une aide à l'élaboration de la commande. Dans le cas particulier de l'enseignement de l'automatique [12], cette méthode peut être une base pour aider l'enseignant à déterminer les erreurs faites par les étudiants. Cependant, bien que nous ayons un début de méthodologie pour déterminer la cause d'un blocage, c'est à l'enseignant ou au concepteur dans le cas général, d'analyser et de comprendre d'où vient l'erreur. Pour la suite de nos travaux, il nous semble intéressant de trouver une méthode automatique d'analyse et d'explication des erreurs. Nous pourrions par la suite imaginer une utilisation sur une plate-forme d'enseignement à distance où l'apprenant pourrait s'auto-former sur la modélisation de la commande sous forme de Grafcet.

VII. References

[1] International Electrotechnical Commission, "*Preparation of function charts for control systems*". Publication 848, 2002.

[2] Wonham W. M., Ramadge P.J., "*On the supremal controllable sublanguage of has given language*", SIAM J Control Optimization, flight 25, n°3, p.637-659, 1987.

[3] Carré-Ménétrier V., Zaytoon J., "*Grafcet: behavioural resulting and control synthesis*", European Journal of Control, vol. 8, n°4, p.375-401, 2002.

[4] Philippot A., Tajer A., Gellot F., Carré-Ménétrier V., "*Méthodologie de modélisation dans le cadre de la synthèse formelle des SED*", CIFA'04, 2004.

[5] Marangé P., Tajer A., Gellot F., Carré-Ménétrier V., *"Synthesis of supervised controller based on Boolean constraints and Boolean automata"*, INCOM'2006 : 12th Symposium on Information Control Problems in Manufacturing, Vol. 1, p299-304, St Etienne, may 17-19, 2006.

[6] Chandra V., Kumar R., "*A Discrete Event Systems Modelling Formalism Based one Event Occurrences Rules and Precedences*", IEEE Transactions one Robotics and Automation, vol. 17, n°6, 2001.

[7] Balemi S., G.J. Hoffmann, P. Gyugyi, H. Wong-Toi, G.F. Franklin, "*Supervisory control of a rapid thermal multiprocessor*", IEEE Transactions on Automatic Control, vol. 38, n°7, p. 1040-1059, 1993.

[8] Wang Y., "*Supervisory Control of Boolean Discrete-Event Systems*", M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, June 2000.

[9] Tajer A., *"Contribution aux approches formelles de synthèse de commande spécifiée par Grafcet"*, Thèse de doctorat de l'Université de Reims Champagne Ardenne, novembre 2005.

[10] Roussel J.M., Medina A., Faure J.M., *"Synthèse d'un programme de commande d'un système logique à partir de l'expression algébrique de ses spécifications"*, Actes MSR'03, pages 77-93, Metz , 6-8 octobre 2003.

[11] Rohée B., Riera B., Carré-Ménétrier V., Roussel J-M., *"A methodology to design and check a plant model"*, 3rd IFAC Workshop on Discrete-Event System Design DESDes'06, Rydzyna Castle (Poland), September 25-28, 2006.

[12] Marangé P., Gellot F., Chemla J.P., Riera B., *"Requierement and Use for remote teaching of Discrete Events Systems"*, Proceeding of 7th IFAC symposium on Advances in control Education, ACE'06, Paper WeP02.1, CD-ROM, Madrid, Spain, June 21-23, 2006.