MODELISATION DES SYSTEMES EMBARQUES LOGIQUES POUR LE DIAGNOSTIC EMBARQUE

Ramla Saddem, Abdoul Karim Armand Toguyeni

10ème Conférence Francophone de Modélisation, Optimisation et Simulation- MOSIM'14 – 5 au 7 novembre 2014 - Nancy –France « de l'économie linéaire à l'économie circulaire»

MODELISATION DES SYSTEMES EMBARQUES LOGIQUES POUR LE DIAGNOSTIC EMBARQUE

R. SADDEM

CRESTIC / Université de Reims Champagne-Ardenne
BP 1039 - 51687 REIMS Cedex 2 - France
ramla.saddem@univ-reims.fr

A.K.A. TOGUYENI
LAGIS / Ecole Centrale de Lille
BP 48, Cité Scientifique
59651 Villeneuve d'Ascq - France
armand.toguyeni@ec-lille.fr

RESUME : *Dans ce papier, nous proposons une nouvelle méthode d'ingénierie permettant la construction systématique du modèle de systèmes embarqués logiques pour le diagnostic de leurs fautes avec l'approche diagnostiqueur. Cette méthode est basée d'une part sur une décomposition structurelle du système en composants logiques et d'autre part sur une abstraction comportementale et une abstraction temporelle. La méthode proposée permet de transformer le comportement d'un système modélisé sous la forme de machine de Mealy en un automate temporisé permettant de construire un diagnostiqueur.*

MOTS-CLES : *Systèmes embarqués Logiques, Machine à états, méthode de réduction, transformation de modèles, diagnostiqueur.*

1 INTRODUCTION

Les Systèmes Embarqués Logiques (SEL) sont des systèmes proposant des fonctions logiques mises en œuvre à l'aide de composants numériques (CPLD, FPGA, microcontrôleurs). La croissance du marché du numérique et la baisse du coût de ces composants permet leur utilisation pour le développement des applications embarquées. Nous nous intéressons plus particulièrement aux systèmes embarqués critiques utilisés pour la commande de systèmes de transport. Leur criticité résulte du caractère sécuritaire des fonctions contrôlées (Par exemple le système de freinage d'un train). Elle exige donc la conception d'architectures tolérantes aux fautes permettant au système de poursuivre sa mission en mode de fonctionnement dégradé mais sécurisé. Cela nécessite de diagnostiquer en ligne les fautes même des composants du système de commande. Notre approche de diagnostic est inspirée de la technique du diagnostiqueur de Sampath (Sampath et al, 1996), mais elle s'appuie sur le formalisme des automates temporisés. Aussi l'objectif de cette modélisation est de construire pour chaque composant du SEL, un modèle de taille réduite permettant d'éviter l'explosion combinatoire lors de la synthèse du diagnostiqueur.

L'article est structuré comme suit. Dans la section 2, nous présenterons le contexte de l'étude. Nous définirons notamment les SEL et nous spécifierons les exigences de surveillance des SEL. Dans la section 3, nous présenterons l'approche composant. En section 4, nous proposerons une nouvelle méthode de réduction du modèle comportemental d'un composant logique (CL). C'est une des contributions majeure de ce travail. En section 5, nous proposerons une méthode permettant la transformation du modèle comportementale d'un CL en automates temporisés (AT).

2 CONTEXTE DE L'ETUDE

2.1 Les systèmes embarqués logiques

Les systèmes embarqués (SE) sont des systèmes électroniques programmables basés sur des composants numériques programmables. Un système logique, est un système qui interagit avec son environnement par l'intermédiaire d'entrées/sorties tout ou rien. Ces systèmes implémentent des fonctions logiques qui peuvent être soient purement combinatoires ou séquentielles. Chaque fonction logique (FL) permet de calculer au fil du temps la valeur d'une sortie en fonction de ses entrées et de son état interne (pour les systèmes séquentiels). Les SEL sont donc à la fois des SE et des Systèmes à Evénements Discrets. A ce titre, leur comportement discret peut être modélisé par les formalismes de systèmes de transitions comme les automates à états finis ou les automates temporisés, voire les réseaux de Petri. Dans cette étude nous avons choisi de les modéliser sous forme d'automate temporisé afin d'appliquer une approche de synthèse de diagnostiqueur inspirée de l'approche de Sampath (Sampath et al, 1996).

2.2 Les exigences de surveillance d'un SEL

D'un point de vue fonctionnel, la surveillance des systèmes auxquels nous nous intéressons a d'abord pour priorité de garantir le bon fonctionnement du système considéré lors de son exploitation. C'est par exemple le cas des cartes de contrôle-commande à base de composants numériques utilisés dans les systèmes de transport. Elles disposent en général de deux catégories de sorties :

les sorties normale et les sorties critiques. Une sortie est dite critique, si elle sert à commander un actionneur participant à une fonction critique de l'automatisme. Aussi, le premier objectif du système de surveillance est donc de déterminer si une sortie a la bonne valeur compte tenu de l'état du système, et si elle garde cette valeur conformément à cet état. Dans ce cadre, on doit surveiller si :

- O1 : une sortie critique doit changer de valeur et reste collée à sa valeur courante,
- O2 : une sortie critique doit conserver sa valeur courante et change intempestivement de valeur.

Le deuxième objectif de la surveillance est de garantir le bon fonctionnement du système surveillé à sa mise en exploitation. En effet, les systèmes complexes sont souvent conçus en prenant en compte les taux de défaillances conditionnels des composants pour en déduire leurs probabilités de pannes.

Le troisième objectif de la surveillance est de réduire les durées d'indisponibilité des systèmes en facilitant leur maintenance. Cela passe notamment par l'identification rapide des composants en panne afin de les réparer ou de les remplacer. Cet objectif est moins prioritaire d'un point de vue sécurité mais tout aussi indispensable sur le plan économique.

3 APPROCHE COMPOSANTS ET MODELISATION COMPORTEMENTALE

3.1 Décomposition d'un système en composants

Les systèmes complexes ne sont pas simple à modéliser vu le grand nombre d'éléments qui les composent et les relations entre ces éléments. Cependant, on peut représenter un système en sous-systèmes. Cette opération peut être renouvelée pour chaque sous-système jusqu'à atteindre les composants élémentaires. Le plus bas niveau de décomposition d'un sous-système est le composant élémentaire.

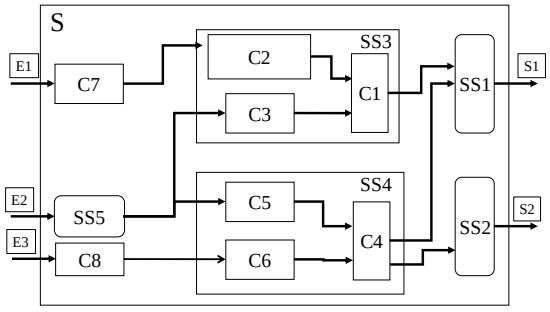

Figure 1 : Flux entre les différents constituants d'un système logique

La Figure 1 montre un système décomposé en sous-systèmes et composants. Il comprend cinq sous-systèmes notés SS_i, $i \in [1..5]$, des composants nommés C_7 et C_8, ainsi que des ports d'entrées notés E_n, $n \in \{1,2,3\}$ et des ports de sorties nommés S_k, $k \in \{1,2\}$. Les sous-systèmes sont eux-mêmes décomposés en sous-systèmes et/ou composants. Par exemple, SS_3 est décomposé en trois composants C_1, C_2 et C_3. Afin de faciliter la désignation des composants nous utiliserons la notation $SS_i.C_j$ pour désigner le composant C_j du sous-système SS_i.

3.2 Le concept de composant logique

Dans cette étude, nous sommes plus particulièrement intéressés aux systèmes embarqués communiquant avec leur environnement à l'aide de nombreuses entrées/sorties logiques (SEL). Le cœur de ce type de système est souvent constitué par un composant numérique de type microcontrôleur ou FPGA. Le comportement de tels systèmes est représenté par des systèmes états/transitions. Il caractérise la manière dont le système transforme ses entrées pour produire ses sorties. Cependant, la présence d'un nombre important d'entrées/sorties entraine un risque d'explosion combinatoire, notamment dans le cadre de formalismes comme les automates à états finis. Pour éviter cet inconvénient, nous proposons de décomposer chaque constituant en un ensemble de composants logiques (CL).

Comme illustré par la Figure 1, chaque constituant d'un système possède des entrées lui permettant de recevoir un flux issus d'autres constituants et des sorties lui permettant d'initier des flux vers d'autres constituants. Par exemple le constituant SS_4 possède deux entrées et deux sorties. Il est en interaction avec SS_5 et C_8 qui lui envoient des flux d'entrée. Il traite ces flux en utilisant le comportement de ses composants internes (C_4, C_5 et C_6) et génère des flux en sortie. Il émet un flux vers SS_1 et un autre flux vers SS_2.

Etant donné une sortie de la carte, comme la sortie S_2, nous désignons par ligne fonctionnelle l'ensemble des composants utilisés pour fournir le flux de sortie correspondant. Ainsi la ligne fonctionnelle de S_2 comprend les constituants S_2, SS_2, $SS_4.C_4$, $SS_4.C_5$, $SS_4.C_6$, C_8, SS_5, E_3 et E_2. En l'absence de redondance (comportement de type OU), la panne d'un de ces constituants conduit à une rupture de flux et donc à une faute du système.

Définition 1 : Composant logique de base

Un CL de base est un composant caractérisé par une unique sortie logique et une ou plusieurs entrées logiques. Il est formellement défini par une FL reliant sa sortie à ses entrées. S'il a un comportement séquentiel, il possède un état interne défini par une fonction de succession.

3.3 Illustration de l'approche composant : décomposition structurelle d'une carte de commande tolérante aux fautes

Nous allons illustrer notre approche sur l'exemple d'une carte de commande du système de freinage d'un train (Johansson, 2004). Afin de simplifier la

présentation de notre méthode, nous limiterons notre carte SEL à trois signaux T.O.R (Tout ou rien) en entrées et deux signaux T.O.R. en sorties. Elle est constituée de trois répartiteurs notés R_j, $j \in [1..3]$, trois cartes filles notées DC_i ,$i \in [1..3]$ et deux voteurs notés V_k, $k \in [1..2]$ (Figure 2).

La fonction d'un répartiteur est de répartir un signal d'entrée entre les différentes cartes filles. Ainsi, un répartiteur peur être représenté comme un système logique à une entrée er_j et trois sorties sr_j^i, l'exposant *i* indiquant la carte fille à laquelle est destinée la sortie (Figure 3a). Nous faisons l'hypothèse ici qu'une faute d'un répartiteur affecte ses trois sorties.

Chaque carte fille *DCi* est conçue autour d'un composant programmable disposant d'entrées/sorties, d'une entrée d'horloge pour la synchronisation de son comportement et d'une entrée « reset » pour son initialisation. La carte dispose par ailleurs, de ports d'entrées (trois ici, notés e_i^j) et de ports de sorties (deux ici, notés S_i^k) lui permettant de recevoir en entrée les signaux issus des répartiteurs et de fournir des sorties destinées aux différents voteurs. Nous supposons par ailleurs que les constituants internes du composant programmable mettent en œuvre une fonction $F_i^k = f_k(\vec{e_i})\ avec\ \vec{e_i} = \sum_{j=1}^{n} \vec{e}_i^j$ permettant de générer le signal de la sortie S_i^k, avec *i* l'indice de la carte fille et *k* l'indice du voteur concerné par la sortie (Figure 2b).

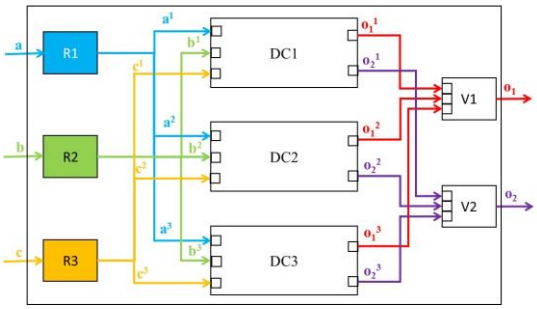

Figure 2 : Synoptique de l'architecture générale de la carte de commande

Nous supposons que chaque entrée et chaque sortie de chaque fonction de carte fille peut être affectée par une faute. De même chaque fonction interne F_i^k d'un composant programmable peut être affectée par une faute que nous supposerons ici la même quel que soit le constituant interne cause de la faute.

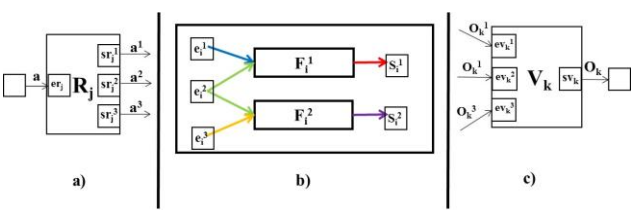

Figure 3 : Spécification des entrées/sorties de la carte de commande

Remarque : Dans ce travail nous supposons que chaque carte fille dispose de sa propre horloge. Cette contrainte matérielle entraine que les sorties des cartes envoyées vers un voteur sont asynchrones. Cet asynchronisme induit une contrainte de modélisation supplémentaire qu'il faudra prendre en compte.

Le troisième type de constituant de la carte de commande que nous considérons est le type voteur. Pour notre application, la carte de commande est sensée mettre en œuvre des votes 2 sur 3 (2oo3). Donc chaque voteur possède trois ports d'entrées ev_k^i avec *i* l'indice de la carte fille auquel est connecté le port, et un port de sortie sv_k. Chaque port d'entrée d'un voteur peut être en faute indépendamment des autres ports. Le port de sortie peut également être en faute.

Du point de vue d'une carte fille, une ligne fonctionnelle comprend l'ensemble des composants qui permettent de mettre la sortie S_i^k à 1 ou à 0 compte tenu du vecteur d'entrées pris à une date donnée. Par exemple, si on se réfère aux Figure 2 et Figure 3b, la ligne fonctionnelle de S_i^1 comprend les composants $er_1, er_2,\ sr_1^i, sr_2^i, e_i^1, e_i^2, F_i^1, et\ s_i^1$. Du point de vue du voteur V_1, la ligne fonctionnelle de sa sortie comprend les trois lignes fonctionnelles des trois sorties des cartes filles qui alimentent ce voteur, plus les composants du voteur $sv_1,\ ev_1^1, ev_1^2, ev_1^3$.

L'ensemble de ces composants sont susceptibles d'être en faute. La détection revient à dire si une ligne fonctionnelle est défaillante. Le diagnostic consiste à identifier le composant responsable de la faute.

3.4 Modèle générique d'un port d'entrée ou de sortie

Un port de constituant peut être vu comme un système à retard permettant de transformer un signal d'entrée en un signal de sortie (Figure 4). Nous proposons donc d'abstraire le comportement de tous les ports d'entrée ou de sortie du système par le schéma de la Figure 5a. Cette figure montre qu'un signal d'entrée se caractérise par des changements de valeurs entre 0 et 1.

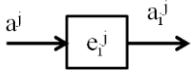

Figure 4 : Schéma générique d'un port d'E/S

La Figure 5b schématise la manière dont nous allons désormais représenter cette abstraction. Nous noterons ra^j le front montant sur le signal d'entrée a^j et fa^j sont front descendant. Lorsque le signal en entrée du port change d'état (par exemple observation de ra^j), le comportement normal du port consiste à générer un signal en sortie (par exemple ra_i^j) dans un délai compris entre 1 unité de temps (ut) et 10 ut. Si au bout de 10 ut, il n'y a pas génération de ce signal, cela correspond à une faute de e_i^j que nous noterons de_i^j.

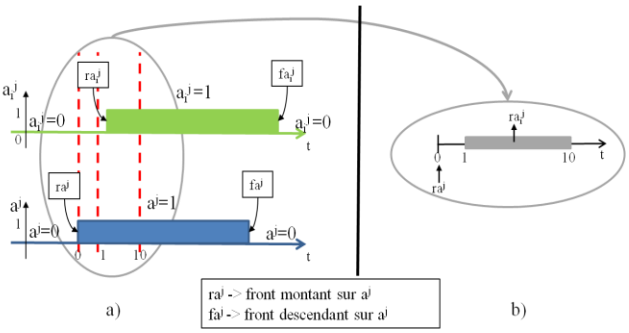

Figure 5 : Abstraction temporelle du comportement d'un port d'E/S

Compte tenu de cette abstraction nous proposons le modèle générique de la Figure 6 modélisant le comportement complet du port. C'est un automate temporisé qui modélise le passage d'un signal du niveau bas au niveau haut et inversement. Les états 0 à 3 modélisent le comportement normal et les états 4 et 5 celui défaillant. Nous supposons qu'initialement le composant est dans l'état bas (i.e. l'état 0). Dans l'état 0, nous avons $a^j = 0$ *et* $a_i^j = 0$. Sur l'occurrence de l'événement ra^j nous commutons vers l'état 1 en initialisant l'horloge x. Si le signal de sortie passe de l'état bas à l'état haut dans un délai de 1 ut à 10 ut, le système commute de l'état 1 vers l'état 2. La borne supérieure du délai de commutation est définie par l'invariant $[x \leq 10]$ associée à l'état 1 et la borne inférieure est modélisée par la garde $[x \geq 1]$ de la transition de l'état 1 vers l'état 2. Le comportement dual pour passer de l'état haut à l'état bas se modélise de manière analogue.

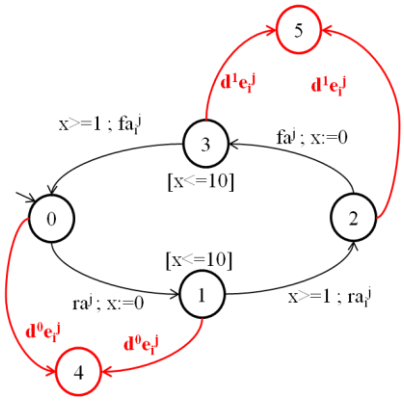

Figure 6 : Modèle générique du comportement complet (normal et défaillant) d'un port d'entrée ou de sortie

Compte tenu de l'exigence de surveillance sur les sorties du système, nous distinguons deux catégories de fautes de port : le collage à 0 que nous noterons $d^0e_i^j$ et le collage à 1 que nous noterons $d^1e_i^j$. Le collage à 0 peut survenir alors que le signal d'entrée du port est à l'état bas (transition de de l'état 0 à l'état 4) ou en cours de fonctionnement du port (transition de l'état 1 vers l'état 4 sur l'occurrence de la faute $d^0e_i^j$). La prise en compte du collage à 1 s'effectue de manière analogue et amène à ajouter l'état 5 (Figure 6).

4 ABSTRACTION DE COMPORTEMENT D'UNE FONCTION LOGIQUE PAR REDUCTION D'UN GRAPHE D'ETATS

A présent nous allons nous intéresser à la modélisation du comportement d'une FL correspondant au comportement d'un CL de base. Les étapes de cette modélisation sont résumées par la Figure 7.

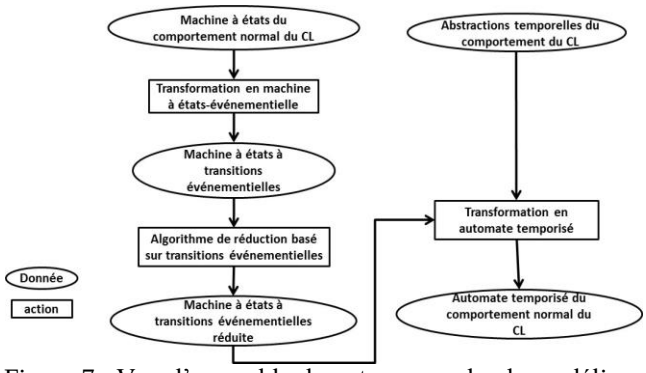

Figure 7 : Vue d'ensemble de notre approche de modélisation

La première partie de la méthode de modélisation d'une FL est une abstraction comportementale permettant la réduction de taille du modèle comportemental initial de la FL. Cette partie est constituée de deux étapes. Elle est présentée dans cette section. La deuxième partie de la méthode consiste en une étape de transformation de la machine d'états à transitions événementielles gardées obtenues précédemment et en un enrichissement de ce modèle à l'aide d'une abstraction temporelle du comportement de la FL. On obtient au final un automate temporisé qui est une abstraction du comportement de la FL pour la synthèse d'un diagnostiqueur. Cette deuxième partie fera l'objet de la section 5.

Dans cette partie, nous commencerons par rappeler succinctement les principales méthodes de réduction des systèmes logiques. Ensuite, nous proposerons une abstraction de comportement basée sur la réduction d'un graphe d'états à transitions événementielles gardées. Ce modèle correspond à un nouveau concept de modélisation que nous proposons pour la mise en œuvre de cette étape d'abstraction comportementale.

4.1 Méthodes de réduction des systèmes logiques : état de l'art

Lors de la synthèse d'un système logique, on utilise souvent des méthodes de réduction pour réduire la taille du modèle de comportement et donc réduire le nombre de composants nécessaires pour sa fabrication. La réduction consiste en la suppression d'états qui sont équivalent dans une machine à états. Les trois méthodes principales pour la réduction d'états sont (Whitaker, 1995) : la méthode de ligne correspondante « row matching », la méthode de la table des implications « implication charts », et la méthode de partitionnement successif « successive partitioning ».

La méthode de ligne correspondante est la plus simple de ces trois méthodes. Elle fonctionne bien quand on a des tables de transition d'état qui ont un état suivant évident et des équivalences entre les sorties. Cette méthode ne donne pas en général la machine à états la plus optimale d'un point de vue réduction. Mais sa simplicité fait qu'elle est souvent utilisée. La méthode de la table des implications utilise une table pour trouver les états équivalents et ainsi réduire la taille de la machine à états. La méthode du partitionnement successifs est quasiment une méthode hybride entre les deux autres méthodes. Elle utilise la table graphique et l'équivalence d'incidence. Ces trois méthodes n'aboutissent pas forcément au même modèle réduit. Pour chaque méthode, la qualité de la réduction dépend de la structure du modèle initial. Ces méthodes ont été conçues pour la réduction de nombre de portes logiques nécessaires pour la mise en œuvre d'un système logique. Elles ne sont pas adaptées à notre méthode de diagnostic dans la mesure où l'objectif de réduction n'est pas le même. Par exemple, l'application de la méthode de la table d'implications à l'exemple de la Figure 8 permet de fusionner les états 4 et 7 de l'automate car ils sont équivalents. Dans la suite de ce papier, nous allons donc proposer une nouvelle méthode de réduction adaptée à nos objectifs de modélisation pour la surveillance des SEL. Cette méthode permettra de réduire à deux états cet exemple.

4.2 Réduction de comportement pour le diagnostic des SEL

4.2.1 Modélisation du comportement d'une fonction-logique

La difficulté pour la modélisation du comportement d'une telle fonction réside dans le fait que le modèle dépend du code implémenté dans le composant programmable. Nous chercherons toutefois à proposer une méthode de modélisation qui soit assez générique et qui permette de construire rapidement le modèle sous-jacent en fonction du code implémenté. Nous supposons que le comportement de la fonction peut être abstrait sous la forme d'un modèle états/transitions de type automate.

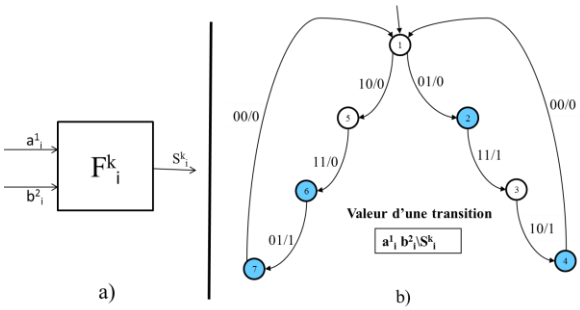

Figure 8 : Exemple de machine de Mealy (b) pour un composant logique à deux entrées (a)

Afin d'illustrer la méthode que nous proposons, nous supposons que nous avons un composant implémentant le comportement séquentiel modélisé par la machine de Mealy donnée par la Figure 8. En effet, nous pouvons remarquer que la valeur de la sortie ne dépend pas seulement du vecteur d'entrées mais également de l'état interne du système. Par exemple, on voit que lorsque nous sommes dans l'état 2, le vecteur d'entrées $\vec{e_t} = (1|1)^T$ donne la valeur 1 en sortie. Par contre, le même vecteur d'entrées nous donne 0 en sortie lors d'une transition de l'état 5 vers l'état 6.

4.2.2 Graphe d'états à transitions événementielles gardées

Pour réduire le comportement d'un composant logique, nous proposons la notion de graphe d'états à transitions événementielles gardées. En effet, dans les représentations classiques des systèmes logiques séquentiels (Machine de Mealy ou machine de Moore), les états modélisent les différentes valeurs de la sortie d'un composant suite à un changement du vecteur d'entrées et les transitions modélisent ces changements d'états. Nous transformons une machine de Mealy en un graphe d'états à transitions événementielles gardées.

Pour modéliser les systèmes logiques asynchrones, il est traditionnellement admis que des entrées non corrélées ne peuvent pas être synchrones (David et Alla, 1992). C'est-à-dire que deux entrées ne peuvent pas changer de valeur en même temps. Cela implique que chaque modification d'une composante du vecteurs d'entrées peut être modélisé par un *ET* logique entre une équation booléenne caractérisant l'état des variables n'ayant pas changées de valeurs et un événement relatif à l'entrée logique ayant changé de valeur. Nous appelons garde, l'équation logique associée à un évènement de transition. Ainsi la transition entre un état S_i et un état S_j est réalisée si l'évènement de transition (modification de l'entrée n) survient et que l'équation booléenne des n-1 autres variables d'entrée est vraie (la garde est vérifiée). Ce concept de garde est inspiré de celui utilisé par le formalisme d'automate temporisé d'UPPAAL (Larsen et al., 1997).

Considérons l'exemple de la Figure 8 et notamment le vecteur d'entrées $\vec{e_t} = (1|0)^T$ permettant de faire passer le CL de l'état 1 vers l'état 5. Dans l'état 1, les deux entrées sont à l'état bas (cf. sur la Figure 8, les transitions des états 4 ou 7 vers l'état 1). Cela signifie donc que dans l'état 1, le vecteur d'entrées vaut $\vec{e_t} = (0|0)^T$. On voit donc que la composante a_i^1 est passée de la valeur 0 à la valeur 1. Cette transition correspond donc à un front montant sur l'entrée a_i^1 (ra_i^1). Notons que cette transition ne modifie pas la valeur de la sortie S_i^k qui reste à 0. Par contre, la transition de l'état 2 vers l'état 3 correspond au même évènement mais cette fois-ci elle fait changer la valeur de la sortie qui passe de 0 à 1. Si on fusionne les états 1 et 2 dans un même macro-état, il faut différentier ces deux transitions car elles ne conduisent pas au même

état. Un moyen consiste à prendre en compte la valeur des autres composantes du vecteur d'entrées lors de l'occurrence de l'évènement. Ainsi nous pouvons constater sur la Figure 9, que cela conduit à prendre en compte que dans la transition de l'état 1 à l'état 5, $b_i^2 = 0$ alors que lors de la transition de l'état 2 vers l'état 3, $b_i^2 = 1$. Nous notons donc les deux transitions respectivement $< 1 > \xrightarrow{b_i^2=0 * ra_i^1} < 5 >$ et $< 2 > \xrightarrow{b_i^2=1 * ra_i^1} < 3 >$.

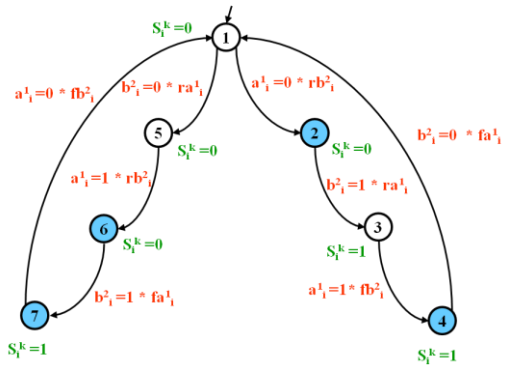

Figure 9 : Graphe d'états à transitions évènementielles gardées correspondant à l'automate de la Figure 8

Dans le reste de ce papier, la relation qui résume le changement d'une des composantes du vecteur d'entrées d'un CL est appelée «événement gardé ».

4.2.3 Méthode de réduction

Cette méthode a pour objectif d'optimiser la réduction tout en respectant les exigences du diagnostic. Elle correspond à l'algorithme de réduction donné en annexe 1. Nous allons présenter ici le principe de cette méthode aux travers de règles. Mais avant cela, nous introduisons les principaux concepts que nous utiliserons dans cette présentation (transition de valeur, transition d'évolution, état de référence).

Définition 2: Transition de valeur

Une transition de valeur est une transition dans le modèle à base d'évènements qui modifie la valeur de la sortie. Le vecteur d'entrées associé à une transition de valeur est appelé vecteur d'entrées de référence.

Définition 3: Transition d'évolution

Une transition d'évolution est une transition dans le modèle à base d'évènements qui permet d'évoluer dans les états internes, mais qui maintient la valeur de la sortie de l'état précédent.

Définition 4: Un état de référence

Un état de référence est l'état origine d'une transition de valeur.

A titre d'exemple, dans la Figure 9, les états 2, 4, 6 et 7 sont des états de référence (états avec fond bleu).

Notre algorithme de réduction est basé sur les 7 règles suivantes.

Règle 1 : Partant d'un état, regrouper dans un même macro-état, tous les états qui ont la même valeur et sont accessibles uniquement avec des transitions d'évolution.

Règle 2 : Pour chaque transition d'évolution, ajouter une transition boucle sur le macro-état.

Règle 3 : Pour l'état atteint par une transition de valeur, créer un macro-état et appliquer la Règle 2.

Règle 4 : Pour chaque transition de valeur, créer une transition entre le macro-état contenant l'état de référence de la transition et le macro-état contenant l'état atteint par la transition.

Règle 5 : Si en appliquant la Règle 1, une transition d'évolution lie deux macro-états, alors regrouper ces deux macro-états.

Règle 6 : Si deux macro-états possèdent la même valeur de sortie et ont les mêmes macro-états précédents et les mêmes macro-états suivants par des transitions de valeur, alors regrouper ces macro-états.

Règle 7 : Si deux macro-états ont la même valeur de sortie et sont précédés par le même macro-état et reliés à ce dernier par deux transitions de valeur marqués par le même événement gardé, alors fusionner ces deux macro-états.

Pour mieux comprendre ces règles, nous les appliquons au graphe d'états de transitions événementielles gardées de la Figure 9. Nous démarrons de l'état 1 qui est l'état initial du modèle comportemental de notre CL. Il existe une transition d'évolution de l'état 1 vers l'état 2 et une autre transition d'évolution de l'état 1 vers l'état 5. Par application de la Règle 1, ces 3 états sont regroupés dans un même macro-état S1 dont la valeur de sortie est égale à 0. Il y a une autre transition d'évolution de l'état 5 vers l'état 6. Donc l'état 6 est ajouté au macro-état S1 (Figure 10a).

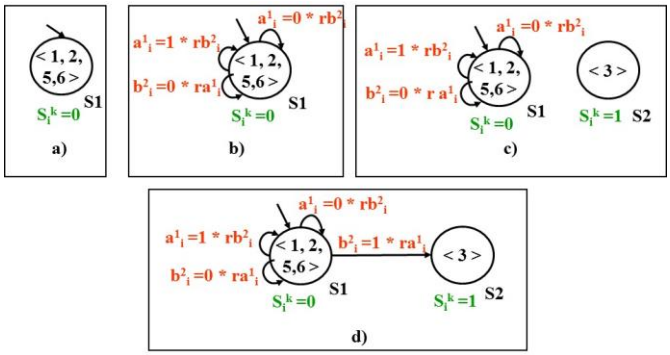

Figure 10 : Application des quatre premières règles de réduction sur l'exemple de la Figure 9.

L'application de la Règle 2 ajoute 3 boucles au macro-état S1, avec les événements gardés des transitions

d'évolution correspondantes. Nous voyons bien sur la Figure 10b les 3 boucles ajoutées à l'état S1. Il existe une transition de valeur de l'état 2 vers l'état 3. D'après la Règle 3, nous créons un macro-état S2 contenant l'état 3 et dont la valeur de sortie est égale à 1 (Figure 10c). La Règle 4 ajoute une transition du macro-état S1 vers le macro-état S2 étiqueté par l'événement gardé (Figure 10d). Après l'application de la Règle 1, nous ajoutons l'état 4 au macro-état S2 avec une boucle modélisant la transition d'évolution de l'état 3 vers l'état 4. Comme il existe une transition de valeur de l'état 4 vers l'état 1, la Règle 4 ajoute une transition du macro-état S2 vers le macro-état S1.

Le même raisonnement nous permet de créer le macro-état S3 qui contient uniquement l'état 7. Une transition étiquetée par l'événement gardé de la transition de valeur de l'état 6 vers l'état 7 lie le macro-état S1 au macro-état S3. Ensuite une autre transition est ajoutée de S3 vers S1 pour modéliser la transition de valeur de l'état 7 vers l'état 1. Le modèle obtenu est représenté par la Figure 11a.

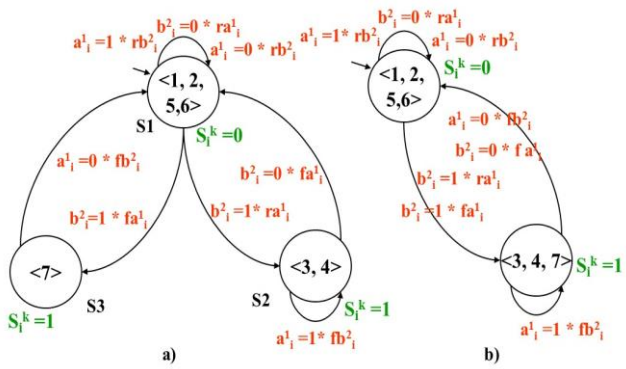

Figure 11 : Application des règles 5 et 6 sur l'exemple de la Figure 9.

La Règle 5 peut être appliquée dans le cas suivant. Supposons que l'algorithme de réduction commence par l'état 5. Ceci conduirait à regrouper les états 5 et 6 dans un macro- état S1. Ensuite, le macro-état S3 serait créé pour l'état 7. Après, en appliquant la Règle 3, un macro-état S0 serait créé à la suite de la transition de valeur de l'état 7 vers l'état 1 et parce que l'état 1 n'est pas encore dans un macro-état. Toutefois, comme il y a une transition d'évolution de l'état 1 vers l'état 5, l'application de la Règle 5 mène à regrouper le macro-état S0 et le macro-état S1 en un seul macro-état S1. Cet exemple nous montre que la Règle 5 est en effet une conséquence de la Règle 1.

Considérons la machine à états réduite de la Figure 11a. Les macro-états S2 et S3 correspondent à la même valeur de sortie 1. Ils ont le même macro-état précédent S1 et le même macro-état suivant S1. Par conséquent, et d'après la Règle 6 ils peuvent être regroupés en un seul macro-état qui représente une abstraction du comportement du composant lorsque sa valeur de sortie est égale à 1. La Figure 11b représente le résultat.

La Règle 7 permet de supprimer l'indéterminisme. Si d'un macro-état, on peut franchir deux transitions de valeur ayant le même événement gardé, on est dans une situation d'indéterminisme qui doit être supprimée. Afin d'assurer le déterminisme de la machine à états réduite basée sur les événements gardés, nous donnons une proposition suffisante.

Proposition 1 : Réduction déterministe

La réduction d'un graphe d'états à transitions évènementielles gardées donne une machine à états déterministe s'il n'y a pas de macro-état qui intègre deux états reliés par une transition d'évolution étiquetée par un événement gardé et un état de référence associé duquel serait issu une transition de valeur associée au même événement gardé et permettant de transiter vers un autre macro-état.

Preuve: Un indéterminisme dans le modèle réduit est produit lorsque depuis un macro-état, nous avons deux transitions différentes étiquetées par le même événement gardé vers deux macro-états différents. Deux cas peuvent avoir lieu :

Cas 1: une transition boucle dans un macro-état S_i avec une transition de valeur de S_i vers S_j.

Si nous avons une transition boucle dans S_i, donc il existe deux états e_k et e_n internes dans S_i qui sont reliés par une transition d'évolution étiquetée par l'événement gardé relatif à cette boucle. S'il existe une transition d'évolution de S_i vers S_j, ceci implique qu'il existe un état e_m interne dans S_i relié à un état e_z interne dans S_j par une transition de valeur étiquetée par l'événement gardé.

Cas 2: une transition de valeur du macro-état S_i vers le macro-état S_j et une autre transition de valeur de S_i vers S_k.

Ce cas est résolu par la Règle7 et ne peut donc pas être présent dans le modèle réduit final.

5 CONSTRUCTION DU MODELE D'UNE FONCTION LOGIQUE POUR LE DIAGNOSTIC D'UN SYSTEM EMBARQUE LOGIQUE

A partir des exigences du diagnostic, notre objectif est de construire un observateur de taille minimale permettant la construction d'un diagnostiqueur répondant aux exigences du diagnostic en ligne d'un SEL. La méthode consiste d'abord à transformer le graphe d'états réduit à transitions événementielles gardées en un automate à états. Cet automate est enrichi à partir d'une abstraction temporelle du comportement de la FL pour obtenir un automate temporisé modélisant le comportement de la FL.

5.1 Transformation de modèles pour la construction systématique de l'automate temporisé du comportement normal d'un composant logique

Nous souhaitons construire l'automate temporisé qui représente le modèle comportemental temporel du CL. Pour cela, nous prenons en compte deux abstractions : le graphe d'états à transitions événementielles gardées et une abstraction temporelle du comportement logique.

L'abstraction temporelle est un moyen de caractériser la performance du CL. Nous supposons ici qu'étant donné un état interne permettant un changement de valeur en sortie d'un CL, une modification du vecteur d'entrées entraine ce changement dans un intervalle de temps compris entre une borne minimale et une borne maximale. Afin d'illustrer cela, considérons de nouveau l'exemple de la carte de commande de la Figure 2. Pour cette carte, nous supposerons que des changements du vecteur d'entrées entrainent un changement de la valeur de sortie de la FL entre 20 et 90 ut. Cette spécification comprend le temps de propagation des signaux d'entrées dans la carte fille, leur traitement par le composant programmable et la propagation du résultat jusqu'à la sortie.

La première étape de la transformation de modèle consiste en la transformation du graphe d'état à transitions gardées en automate à états (Figure 12).

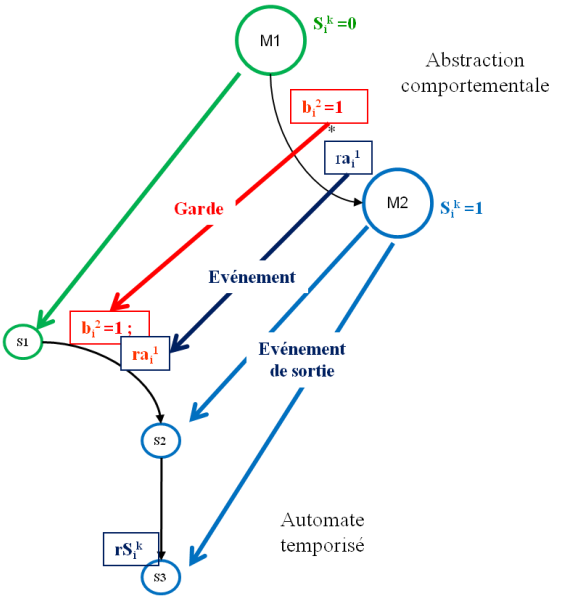

Figure 12 : Etape 1 de la transformation de modèles

Chaque transition de valeur est modélisée par deux transitions. La première transition (comme par exemple la transition $S_1 \rightarrow S_2$) modélise la modification de valeur du vecteur d'entrées de référence et la deuxième transition ($S_2 \rightarrow S_3$) modélise l'occurrence de l'événement de changement de la sortie. En effet, dans le modèle source, le franchissement d'une transition résulte de l'occurrence d'un événement. Suite à l'occurrence de cet événement, la valeur de la sortie change. Donc, deux actions sont produites : l'événement qui entraine la transition et le changement de valeur de la sortie. L'équation logique de la transition de valeur est traduite par une garde dans la première transition et l'événement lui-même est traduit par une action dans cette transition. Ainsi la transition $M_1 \xrightarrow{g.e} M_2$ avec g la garde et e l'événement est transformée en la transition $S_1 \xrightarrow{g\,;\,e} S_2$ avec juste une adaptation à la syntaxe des automates temporisés. L'état destinataire (M_2) du modèle source est transformé en deux états S_2 et S_3 dans le modèle cible. Nous créons alors la transition $S_2 \xrightarrow{rS_i^k} S_3$ qui modélise le changement de la valeur de la sortie considérée suite au changement du vecteur d'entrées de la FL par rapport à un de ses états de référence.

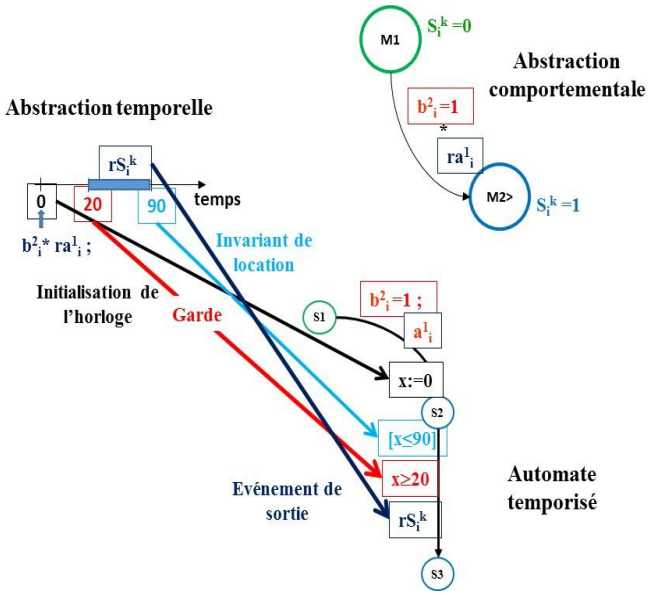

Figure 13 : Etape 2 de la transformation de modèles

La deuxième étape de la transformation consiste à compléter le modèle obtenu lors de l'étape 1 par la prise en compte de l'abstraction temporelle (Figure 13). Nous commençons par initialiser une horloge x dans la première transition pour définir des temps relatifs au changement du vecteur d'entrées. L'hypothèse que la sortie S_i^k doit se produire entre 20 ut et 90 ut après l'occurrence du vecteur d'entrées de référence, est prise en compte par l'invariant de l'état S2 pour la borne maximale, et la garde de la transition de S_2 vers S_3 qui est réécrite en $S_2 \xrightarrow{x\geq 20\,;\,rS_i^k} S_3$.

La transformation du modèle réduit de la Figure 11b et la prise en compte de l'abstraction temporelle, permettent d'obtenir le modèle de comportement normal d'un CL (Figure 14).

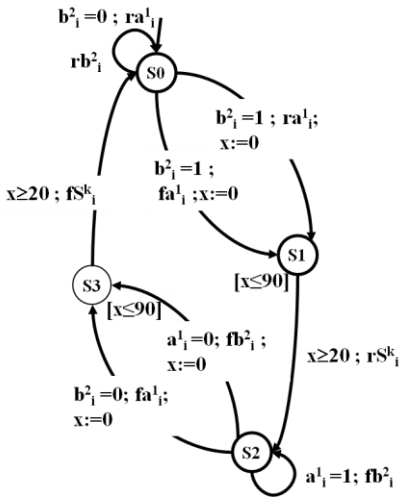

Figure 14 : Automate temporisé résultant de la transformation de modèles

5.2 Obtention du modèle complet d'un composant logique

Selon les exigences du diagnostic, nous pouvons constater que deux fautes peuvent avoir lieu au sein d'un CL affectant sa sortie : $d^1F_i^k$ (respectivement $d^0F_i^k$) qui représente une faute qui empêche le composant de mettre sa sortie à 0 (respectivement à 1). Au modèle de comportement normal nous ajoutons deux états pour représenter les deux états de fautes correspondants : l'état S4 et l'état S5.

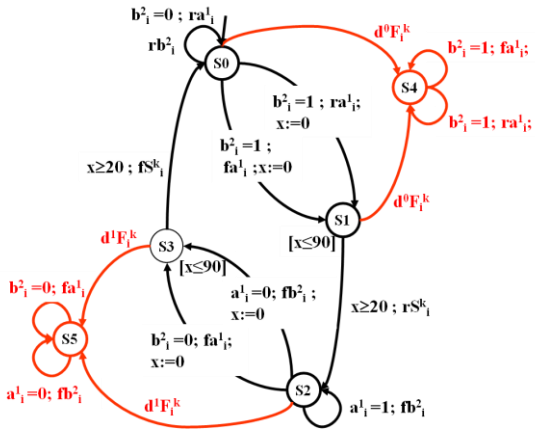

Figure 15 : Modèle de comportement complet d'un composant logique à deux entrées

Ainsi la faute $d^0F_i^k$ peut avoir lieu soit avant l'occurrence du vecteur d'entrées de référence (transition de S0 vers S4), ou bien après son occurrence (transition de S1 vers S4 dans la). Dans les deux cas, la sortie ne passe plus à 1 suite à l'occurrence de cette faute et le composant se retrouve dans l'état S4 (Figure 15). Un raisonnement similaire permet de prendre en compte le deuxième type de faute entrainant des transitions de S2 ou S3 vers S5.

6 CONCLUSION

Dans ce papier, nous avons proposé une nouvelle méthode de modélisation comportementale des SEL pour le diagnostic de leurs fautes avec l'approche diagnostiqueur. Notre approche de modélisation de SEL part d'une décomposition du système en CL. Dans cette étude, nous avons proposé un modèle générique pour modéliser sous forme d'automate temporisé le comportement d'un port d'entrée ou de sortie. D'autre part, nous avons proposé une méthode d'abstraction comportementale permettant de modéliser sous forme d'automate temporisé le comportement d'une fonction logique. L'expressivité des automates temporisés permet d'obtenir un modèle de taille réduite intégrant le comportement séquentiel et les spécifications temporelles du composant logique. L'algorithme de réduction a été mis en œuvre en langage C. Sur des exemples comme celui proposé dans cette étude, nous avons pu vérifier qu'il est correct et efficace.

Les perspectives de ce travail résident d'une part dans le développement d'un outil logiciel pour faciliter la transformation de modèles. Nous pensons nous appuyer sur un outil comme ATL (ATLAS Transformation Language) (Jouault et al., 2008). D'autre part, à partir des modèles ainsi obtenus, nous voulons proposer une méthode de diagnostic basée sur une approche de diagnosticabilité modulaire (Contant et al., 2006) consistant à regrouper des modèles de composants logiques en fonction du degré d'observabilité du module résultant.

REMERCIEMENTS

Ce travail a été en parti subventionné par le Campus International pour la Sécurité et l'Intermodalité dans les Transports, l'Union Européenne et la Région Nord-Pas de Calais. Nous remercions toutes ces institutions.

REFERENCES

Alur R. and D. Dill, 1994. A theory of timed automata, Theoretical Computer Science, vol. 126, p. 183-225.

David R., and H. Alla, 1992. Du Grafcet aux réseaux de Petri. Paris: Hermès.

Johansson R., 2004. *A fault-tolerant architecture for computer-based railway vehicle brake systems.* Journal of Rail and Rapid Transit, Vol. 218, n° 3, p. 189-200.

Jouault F., F. Allilaire, J. Bézivin and I. Kurtev, 2008. ATL: A model transformation tool. Science of Computer Programming, Elsevier, Vol. 72, n°. 1–2, p. 31-39.

Larsen K. G., P. Pettersson, and W. Yi, 1997, Uppaal in a Nutshell. *Journal of Software Tools for Technology Transfer*, Vol. 1, n° 1–2, p. 134-152.

Sampath M., Sengupta R., Lafortune S., Sinnamohideen K. and D. Teneketzis, 1996. *Failure diagnosis using discrete event systems*. IEEE Transactions on Control System Technology, vol. 4, n° 2, p. 105–124.

Whitaker J., 2005. The Electronics Handbook, 2nd Edition, CRC Press, p. 73-74.

ANNEXE 1 : ALGORITHME DE REDUCTION

Notations :
$\Sigma_{\text{nontraité}}$: ensemble des états du modèle non traités
Set_ME : ensemble des macro-états du modèle
ME : Macro-état ; **ME$_C$** : Macro-état courant ; **ME$_i$** : Macro-état initial ; **E** : état ; **E$_C$** : Etat courant
E$_i$: Etat initial ; **F**: file pour contenir les états du modèle à traiter

Début algorithme
Mettre l'ensemble des états du modèle dans $\Sigma_{\text{nontraité}}$
Etape 1 : (* Macro-état initial *)
Créer le macro-état initial (ME$_i$) ; $E_C = E_i$; Ajouter E_i dans ME$_i$; ME$_C$ ← ME$_i$; Ajouter ME$_C$ dans Set_ME
Etape 2 : (* Traitement de tous les états d'un macro-état*)
Etape 2-1 : **Si** la valeur de sortie de ME$_C$ ≠ la valeur de sortie de E$_C$ **alors**

- ME$_C$ ← ME dont la valeur de sortie = la valeur de sortie de E$_C$
- **Si** Ec ∉ ME$_C$ **alors** ME$_C$ ← ME / $Ec \in ME$

Etape 2-2 : **Pour** chaque transition d'évolution t de l'état courant E$_C$ vers l'état suivant E **faire** :

- **Si** E ∈ ME$_C$ **alors** Créer une transition de ME$_C$ sur lui-même correspondant à la transition t
- **Sinon**
- **Si** il existe ME / $E \in ME$ alors Créer une transition de ME$_C$ vers ME
- **sinon**

  Ajouter E à ME$_C$; Enfiler l'état E dans F ; Créer une transition de ME$_C$ sur lui-même correspondant à la transition t
- **fin sinon**
- **fin sinon**

fin Pour

Etape 2-3 : **Pour** chaque transition de valeur t de l'état courant E$_C$ vers l'état suivant E faire :

- **Si** il existe un macro-état dont la valeur de sortie = valeur de sortie de E, soit ME ce macro-état **alors**
  - **Si** E ∈ ME **alors** Créer une transition de ME$_C$ vers ME correspondant à la transition t
  - **sinon** (* Cas ou E n'est pas déjà dans un macro-état *)
    Créer ME' ; Ajouter ME' dans Set_ME ; Ajouter E à ME' ; Enfiler l'état E dans F ; Créer une transition du MEC vers ME' correspondant à la transition t.
  - **fin sinon**
- **sinon (* Cas ou il n'existe pas de macro-état ayant la valeur de sortie de E *)**

  Créer un nouveau macro-état ME dont la valeur de sortie = valeur de sortie de E ; Ajouter ME dans Set_ME ; Ajouter E à ME ; Enfiler l'état E dans F ; Créer une transition du MEC vers ME correspondant à la transition t.
- **fin si**

fin Pour
Etape 2-4 : Retirer E$_C$ de $\Sigma_{\text{nontraité}}$
Etape 2-5 : **Si** F n'est pas vide et $\Sigma_{\text{nontraité}} \neq \varnothing$ **alors**

- E$_C$ ← Défiler F ∩ $\Sigma_{\text{nontraité}}$ (*le nouvel état courant devient le prochain état dans la file qui est non traité*) ; Aller à l'étape 2

Etape 3 : (* Règles d'agrégation *)
TQ non vide (Set_ME) **faire**
ME$_C$ ← Retirer ME de Set_ME

Etape 3-1 : **Si** il existe ME dans Set_ME tel que MEc possède une transition d'évolution t vers un état de ME **alors** Retirer ME de Set_ME ; ME$_C$ ← ME$_C$ ∪ ME ; Créer une transition de ME$_C$ sur lui-même correspondant à la transition t (* self-loop *).
Fin Si
Etape 3-1 : **Si** il existe ME appartenant à Set_ME tel que la (valeur_sortie(ME)=la valeur_sortie(ME$_C$)) et (predecesseur(ME)=predecesseur(ME$_C$)) et ((successeur(ME)=successeur(ME$_C$)) **alors**
Retirer ME de Set_ME ; ME$_C$ ← ME$_C$ ∪ ME ; Créer une transition de ME$_C$ sur lui-même correspondant à la transition t (* self-loop *).
Fin Si
Fin TQ
Mettre à jour les transitions entre les macro-etats
Fin algorithme