UNIVERSITE DE REIMS CHAMPAGNE ARDENNE

THESE

Présentée pour obtenir le grade de

Docteur de l'Université de Reims Champagne Ardenne

Spécialité

GENIE INFORMATIQUE, AUTOMATIQUE ET TRAITEMENT DU SIGNAL

Par

Abdelouahed TAJER

CONTRIBUTION AUX APPROCHES FORMELLES DE SYNTHESE DE COMMANDE SPECIFIÉE PAR GRAFCET

Soutenue publiquement le 3 novembre 2005, devant le jury composé de :

M. Jean-Marc FAURE	Professeur, Institut Supérieur de Mécanique de Paris (ISMEP/SUPMECA)	*Rapporteur*
M. Nidhal REZG	Professeur, Université de Metz	*Rapporteur*
M. Hervé GUÉGUEN	Professeur, Supelec Rennes	*Président de jury*
M. Bernard RIERA	Professeur, Université de Reims Champagne Ardenne	*Examinateur*
Mme. V. CARRÉ-MÉNÉTRIER	Professeur, Université de Reims Champagne Ardenne	*Directrice*
M. François GELLOT	Maître de Conférences, Université de Reims Champagne-Ardenne	*Co-directeur*

Remerciements

Ce travail de recherche a été réalisé au CReSTIC (Centre de Recherche en STIC) de Reims. J'adresse mes plus vifs remerciements à Monsieur Janan ZAYTOON, directeur du CReSTIC, de m'y avoir accueilli.

Je tiens à exprimer particulièrement ma profonde gratitude à Madame Véronique CARRE-MENETRIER, Professeur à l'Université de Reims Champagne-Ardenne, qui a dirigé mes travaux durant ces quatre années de thèse. Ses qualités de chercheur, son obstination et sa patience ont été pour moi, autant d'aides durant ce travail.

Je suis également très reconnaissant envers M. François GELLOT, Maître de conférences à l'Université de Reims Champagne-Ardenne, co-encadrant de cette thèse, pour sa disponibilité et ses nombreux et précieux conseils.

Je remercie Messieurs Jean-Marc FAURE, Professeur à l'Institut Supérieur de Mécanique de Paris (ISMEP/SUPMECA), et Nidhal REZG, Professeur de l'Université de Metz, pour l'honneur qu'ils m'ont fait en acceptant d'être rapporteurs de ce travail. Je leur suis très reconnaissant pour les remarques et suggestions qu'ils m'ont apportées.

Je remercie M. Hervé GUEGUEN, Professeur à Supelec Rennes pour l'honneur qu'il m'a fait en examinant et en présidant ce jury. Mes remerciements vont également à M. Bernard RIERA, Professeur à l'Université de Reims Champagne-Ardenne, pour avoir accepté d'être examinateur de ce mémoire.

J'exprime ma reconnaissance à tous les membres de l'équipe Systèmes à Evènements Discrets du groupe Automatique du CReSTIC, et à toutes les personnes de l'unité LAM du CReSTIC qui m'ont aidé et soutenu au quotidien.

Je n'oublierai pas les amis, dont le soutien moral et affectif, a été de tous les instants.

Enfin, il m'est impossible de dire en quelques mots la gratitude que j'ai pour ma famille. Que ce mémoire soit un témoignage de mon affection indéfectible.

SOMMAIRE

INTRODUCTION GENERALE

Pendant ces dernières décennies, la complexité croissante des Systèmes Automatisés en terme de quantité, de besoins en communication, de diversité des composants, etc., a accru les exigences des utilisateurs en terme de sûreté de fonctionnement et d'aide à la conception de programmes. En effet, jusqu'à présent, le concepteur ne disposait d'aucune aide véritable pour ces deux points. En fait, seule sa compétence et son expérience lui permettaient d'élaborer des programmes dont il garantissait lui-même qu'ils permettent d'obtenir un système sûr de fonctionnement.

Pour élaborer un fonctionnement sûr, deux approches sont possibles : une approche par vérification et une approche par synthèse. Dans l'approche par vérification, la construction de la commande repose sur l'alternance entre une phase de validation, par Model checking [BER 99] par exemple, et une phase de correction des erreurs identifiées. Cependant, plutôt que de se limiter à la vérification et /ou la validation, il est intéressant de développer des techniques permettant la synthèse automatique d'une implantation correcte de la commande.

Ce mémoire présente une démarche formelle permettant la synthèse d'une implantation de la commande à partir d'une spécification Grafcet, d'un système à commander et d'un ensemble de contraintes à respecter. Pour réaliser cet objectif d'implantation, il faut modéliser le système physique que l'on désire contrôler, et ensuite exprimer les objectifs de commande (spécification Grafcet) et finalement utiliser des techniques mathématiques pour élaborer la commande correspondante.

Cependant, les contraintes technologiques relatives à l'environnement d'exécution de la commande sur une cible quelconque (PC ou API), engendrent une distance sémantique entre une spécification exprimée en Grafcet et l'implantation correspondante. Les travaux présentés dans ce mémoire ont donc pour objectif de proposer une démarche formelle de synthèse permettant d'obtenir une implantation correcte et optimale de la commande à partir d'une spécification Grafcet.

Des premiers travaux menés sur la synthèse de la commande spécifiée par Grafcet ont permis de caractériser les étapes nécessaires pour passer d'une spécification Grafcet à l'implantation de la commande correspondante. Pour formaliser ces étapes, il a été nécessaire d'asseoir la démarche sur des outils/méthodes ayant de solides bases formelles, tels que les automates et la théorie de supervision des SED [RAM 89]. L'objectif principal de la théorie de supervision élaborée par Ramadge et Wonham [RAM 89] est de construire des lois de commande par analogie aux systèmes continus en terme de commandabilité, d'observabilité et de commande optimale en utilisant des outils formels de spécifications tels que les langages formels et les automates. Malgré les apports mathématiques solides de cette théorie, elle est peu exploitée dans le cadre d'applications industrielles à cause de la non-convivialité des automates et/ou des langages formels utilisés pour la modélisation de la commande des systèmes complexes.

Dans ce contexte, nos travaux ont abouti au développement d'une approche globale de synthèse d'implantation en associant le Grafcet et la théorie de supervision en vue de la synthèse d'une commande optimale à partir d'un Grafcet, d'un modèle du procédé à commander et des contraintes à respecter sur les évolutions du procédé à commander [CAR 00].

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. Pour pallier ces problèmes, les travaux présentés dans ce mémoire ont pour objectif d'améliorer la praticabilité des travaux précédents en proposant des modèles structurés et évolués pour la partie opérative et les contraintes et intégrant le concepteur dans la boucle de l'élaboration de la commande.

Organisation du mémoire :

Le chapitre 1 présente les notions de bases relatives aux SED ainsi que les fondements de la théorie de supervision selon Ramadge et Wonham. Afin de situer le contexte de nos travaux, différentes approches qui s'intéressent à l'implantation depuis la synthèse de commande par supervision sont ensuite présentées.

Le chapitre 2 présente dans un premier temps les étapes nécessaires pour passer d'une spécification par Grafcet à son implantation dans un environnement API. Ces étapes ont été formalisées par une démarche globale de synthèse en association avec la théorie de supervision [NDJ 99]. Pour montrer les avantages et les inconvénients de la démarche, nous proposons son application sur l'exemple commun développé au niveau du groupe COSED du GDR MACS. Nous avons complété cette approche par une formalisation de la synthèse non plus hors ligne mais en ligne. Enfin, nous situons cette approche par rapport à d'autres approches en terme de modélisation de la partie opérative.

Le chapitre 3 présente une méthodologie de modélisation de la partie opérative à partir des règles d'occurrence et de précédence comme évoquée dans [CHA 02]. Un autre aspect développé dans ce chapitre concerne la modélisation des contraintes avec des équations logiques. Ces nouveaux modèles se substituent aux automates événementiels et rudimentaires utilisés jusqu'à présent dans l'approche de synthèse.

Dans le chapitre 4, nous proposons d'intégrer les modèles présentés au chapitre 3. Pour cela, nous développons une nouvelle démarche de synthèse hors ligne qui tient compte de la différence sémantique des modèles de partie opérative et des contraintes proposés. Dans ce chapitre, nous présentons également comment procurer au concepteur une aide dans l'élaboration de la commande et dans le raffinement des modèles de départ.

Enfin, le chapitre 5 illustre les chapitres 3 et 4 à travers un exemple de commande de tri de caisses.

Chapitre 1

SYNTHESE DE COMMANDE DES SYSTEMES A EVENEMENTS DISCRETS

Dans ce chapitre, nous présentons le contexte de nos travaux de thèse qui portent sur la synthèse de commande des systèmes à événements discrets (SED). Dans une première partie, nous faisons un bref descriptif de la formalisation de la synthèse de commande des SED. Elle trouve son origine dans la théorie de supervision des SED, initiée par Ramadge et Wonham (RW) [RAM 89]. Cette théorie ne fournit pas de méthodologie explicite pour implanter la commande supervisée élaborée, malgré les nombreux travaux qui traitent de ce sujet. La seconde partie de ce chapitre étudie quelques propositions en présentant leurs apports et leurs limites, permettant ainsi de situer le contexte de notre démarche.

1. SYNTHESE DE LA COMMANDE DES SED

1.1. Qu'est ce qu'un SED ?

Un Système à Evénements Discrets (SED) est un système dynamique évoluant suite à l'occurrence d'événements, et dont l'espace des états est discret. Ces événements caractérisent les différents changements physiques qui surviennent dans le système. Leur succession constitue des trajectoires qui permettent de décrire l'espace des états.

Prenons l'exemple d'une machine pouvant occuper trois états différents : repos, panne et occupé. Une évolution possible de cette machine est donnée par la figure 1.1. Dans son état initial, la machine est supposée au repos. Sur occurrence de l'événement α " commencer une opération " à l'instant t_1, la machine évolue vers l'état "occupé". De même, l'occurrence des événements λ "occurrence d'une panne" et γ "réparer la machine" respectivement aux instants t_2 et t_3 conduit la machine vers l'état de panne, pour ensuite revenir vers l'état de repos. A l'instant t_5 par exemple, l'événement β "fin de l'opération" permet de conduire la machine de l'état "occupé" à l'état "repos".

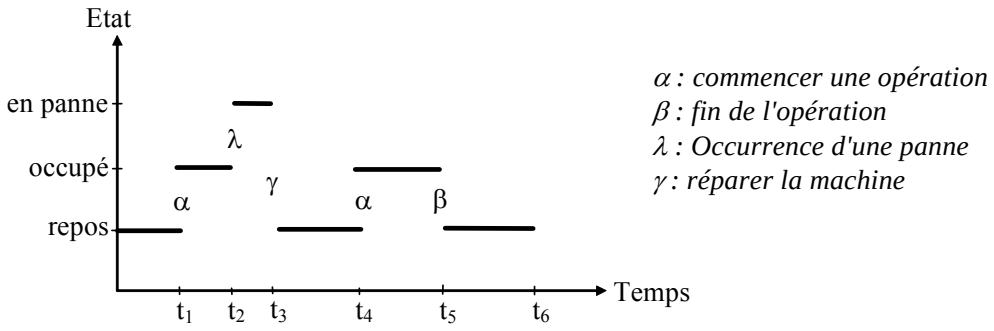


Figure 1.1. Trajectoire décrite par une machine à trois états

Le comportement du procédé peut être décrit par la séquence d'événement $\alpha\lambda\gamma\alpha\beta$. Une telle simplification est justifiée lorsque le modèle est utilisé pour l'étude des propriétés dynamiques indépendamment du temps. Cependant, le comportement du procédé peut être entièrement modélisé par des couples (événement, instant d'occurrence d'événement) de la forme (α, t_1), (λ, t_2), etc. Cette description, appelé modèle temporel, suppose que l'instant d'occurrence des événements est une information pertinente. Cette information peut être représentée par un modèle temporisé non-stochastique (réseau de Petri Temporisé, Algèbre Max, ...) où le temps est connu a priori, ou par un modèle

stochastique (chaînes de Markov, files d'attentes, Processus Semi-Markoviens généralisés, ...) basé sur d'hypothèses statistiques.

Les nombreux domaines d'utilisation des SED ainsi que les différents comportements rencontrés dans chaque domaine, ont conduit au développement d'autres modèles de représentation des SED. Ainsi, l'abstraction de l'aspect temporel dans les occurrences d'événements et la prise en compte uniquement de l'ordonnancement des événements, permettent de définir les modèles de SED dits "logiques" tels les automates. L'exemple de la figure 1.1 serait dans ce cas décrit par la succession des événements α, λ, γ, α et β. Ces modèles logiques servent à l'étude des propriétés dynamiques des événements générés par le système, indépendamment du temps physique. Leur utilisation est assez répandue pour l'étude des propriétés qualitatives des systèmes dans des domaines tels que les protocoles de communication, les protocoles de base de données ou encore l'analyse des circuits intégrés.

Dans les nombreuses applications où les SED ont été utilisés, la formulation et l'analyse du modèle se font de la façon suivante. Dans un premier temps, l'ensemble des phénomènes physiques possibles appelés encore "trajectoires admissibles" est spécifié. Cela est fait par un modèle à états discrets tel que les réseaux de Petri, les automates, le grafcet ou encore les langages formels, etc. Ensuite, il faut s'assurer que pour une propriété donnée (stabilité, non-blocage, vivacité,...), toutes les trajectoires admissibles vérifient la dite propriété.

La théorie de supervision [RAM 89], est considérée comme l'origine de la synthèse de commande des SED. Dans cette théorie, le comportement du procédé en boucle ouverte est séparé du comportement de celui de la commande en boucle fermée. Elle a permis d'établir, pour les SED, des principes analogues à ceux de l'automatique continu, tels que les notions de commandabilité, d'observabilité et de commande optimale, en employant des techniques issues de l'informatique telles que les langages formels et les automates. L'objectif principal de cette théorie est la synthèse d'une commande la plus permissive possible par rapport aux spécifications désirées du fonctionnement tout en évitant le blocage et les états interdits.

1.2. Modélisation pour la synthèse de commande des SED

Dans la théorie de supervision proposée par RW, les contraintes temporelles du système sont ignorées, seul l'ordre et l'identité des événements qui peuvent être générés par le système sont pris en compte. Par conséquent, un modèle logique où seules les séquences d'événements sont manipulées est parfaitement adapté à ce type de problème. L'utilisation des langages formels pour modéliser ce type de système permet d'obtenir un outil mathématique, difficile à manipuler. En effet, un langage formel est défini comme un ensemble de mots de longueur finie déduit d'un certain alphabet fini, et un alphabet est un ensemble de symboles désignant des événements. Un automate à état fini, en étant une représentation graphique plus aisée d'utilisation, est un dispositif qui engendre le langage formel en manipulant son alphabet. Pour plus de détails concernant les modèles de langage formel ou les automates, le lecteur pourra voir par exemple [CAS 99] et [WON 02].

1.3. La théorie de la commande supervisée selon Ramadge et Wonham

Dans le cadre de la théorie de RW, un système à événements discrets est vu comme un procédé générant spontanément des événements. Ce procédé évolue de façon asynchrone, conformément aux occurrences des événements [RAM 89]. Son comportement peut être décrit par un ensemble de séquences d'événements constituant ainsi un langage sur l'ensemble des événements. Par la suite, nous parlerons donc indifféremment de comportement ou de langage.

De façon plus pratique, un procédé G peut être représenté par l'automate de la figure 1.2, qui décrit une machine à trois états : "repos", "occupé" et "panne". Les transitions d'un état à un autre sont supposées instantanées, spontanées et asynchrones.

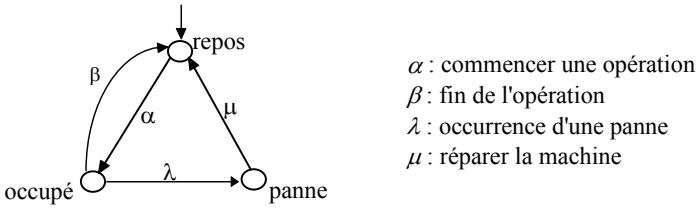


Figure 1.2. Exemple d'un générateur d'événements

Formellement, l'automate de la figure 1.2 est décrit par un quadruplet $G = (\Sigma, Q, q_0, \delta(q, \sigma))$ avec $\Sigma=\{\alpha,\beta,\lambda,\gamma\}$, $Q=\{\text{panne, repos, occupé}\}$, $q_0=\{\text{repos}\}$ et $\delta(q, \sigma)$ la fonction de transition entre chaque état $q \in Q$.

Lorsque le procédé à représenter est très complexe, les travaux autour de la théorie de supervision ont proposé pour le modéliser de le décomposer en sous modules distincts et quasi-indépendants, pour construire ensuite le modèle global par une composition asynchrone des différents sous modules. Imaginons par exemple un petit atelier flexible, constitué de deux machines M_1 et M_2 (figure 1.3) usinant des pièces.

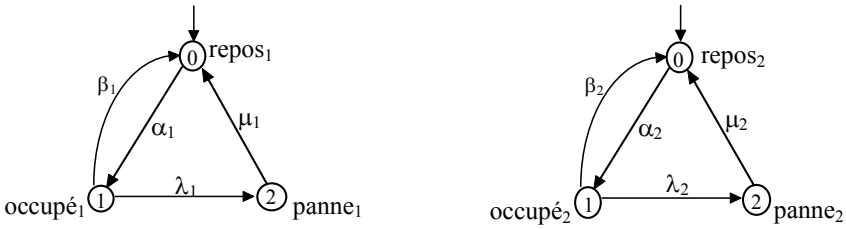


Figure 1.3. Exemple de deux sous systèmes

Les alphabets respectifs des machines M_1 et M_2 sont $\Sigma_1 = \{\alpha_1, \beta_1, \lambda_1, \mu_1\}$ et $\Sigma_2 = \{\alpha_2, \beta_2, \lambda_2, \mu_2\}$, ils permettent de générer les langages L_1 et L_2 décrivant respectivement les comportements de la machine 1 et de la machine 2. Le modèle global de ce système, obtenu par composition asynchrone G1 ✕ G2 [WON 88], est un procédé $G = (\Sigma, Q, q_0, \delta(q, \sigma))$ où $\Sigma = \Sigma_1 \cup \Sigma_2$. Chaque état de Q correspond à un unique couple (q_i, q_j), q_i étant un état de G_1 et q_j un état de G_2, q_{00} correspond au couple (q_0, q_0). δ est une fonction de $Q \times \Sigma \rightarrow Q$.). Pour l'exemple de l'atelier, le procédé global est donné dans la figure 1.4 où la notation q_{ij} sur la figure correspond au couple (q_i, q_j)

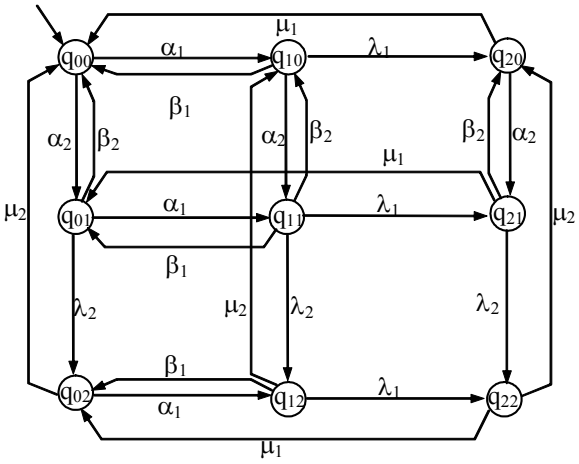


Figure 1.4. Modèle global de l'atelier après produit croisé asynchrone.

1.3.1 Concept de supervision

Le modèle d'un procédé global obtenu par un produit croisé asynchrone, est un générateur spontané de chaînes d'événements, sans contrôle externe. En vue de

commander un tel procédé dans le cadre de la théorie de supervision, certains événements sont interdits ou autorisés en temps voulu. Ainsi, lorsque le procédé se trouve dans un état q_i, il reçoit de la part du superviseur une liste γ^i d'événements autorisés. La prochaine évolution du procédé se fera alors sur occurrence d'un événement $\sigma^{i+1} \in \gamma^i$. A partir du nouvel état q_j atteint par le procédé, une nouvelle liste d'événements autorisés est fournie, et ainsi de suite. Ce principe d'inhibition/autorisation est appelé "supervision" (figure 1.5), et le fonctionnement qui en découle représente le comportement en boucle fermée du procédé.

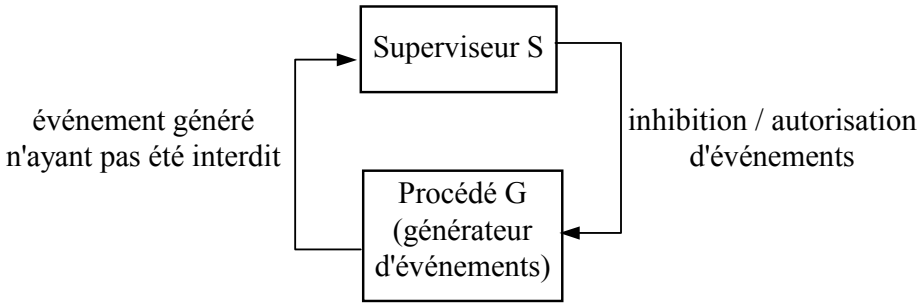


Figure 1.5. Principe de supervision

Pour atteindre cet objectif de supervision, l'ensemble Σ des événements d'un procédé est divisé en deux ensembles disjoints Σ_c et Σ_u, représentant respectivement l'ensemble des événements commandables, et l'ensemble des événements non commandables. Les événements commandables (Σ_c) sont les événements sur lesquels le superviseur peut avoir une action directe, c'est à dire qu'il peut les interdire ou les autoriser à tout instant. Par contre, le superviseur ne peut exercer aucune influence directe sur les événements dits non commandables (Σ_u). Concernant l'exemple de la figure 1.2, les événements commandables sont les ordres de mise en marche et de réparation d'une panne. L'occurrence d'une panne ainsi que l'accomplissement d'une tâche sont considérés comme non commandables. Le superviseur ne pouvant interdire que les événements commandables, tous les événements de Σ_u sont toujours autorisés quel que soit l'événement généré par le procédé, i.e. la condition $\Sigma_u \subseteq \gamma^i$ est toujours vraie pour tout i.

L'action du superviseur consiste à modifier le comportement en boucle ouverte d'un SED afin d'obtenir un langage qui respecte le cahier de charges. Celui-ci représente souvent une limitation du comportement du procédé, généralement spécifiée de manière indirecte à travers différents objectifs qualitatifs de sécurité et de performances.

Le problème qui se pose est alors le suivant : étant donné un langage de spécification, est-il possible de synthétiser un superviseur de telle manière que le procédé G respecte, grâce à des interdictions et des autorisations adéquates, le comportement désiré K. La clé de ce problème se trouve dans la notion de commandabilité.

Un langage K est dit commandable par rapport au langage L(G), lorsque l'occurrence d'un événement non commandable physiquement possible entraîne une évolution vers un état appartenant également à K. La commandabilité d'un comportement K est donc une condition nécessaire et suffisante à l'existence d'un superviseur tel que couplé au procédé G, le fonctionnement en boucle fermée soit K.

Si K n'est pas commandable, alors on doit chercher un sous-ensemble de K appelé C (K) qui a la propriété de commandabilité en respectant toutes les contraintes de sécurité et de vivacité. Cette méthode de recherche est appelée synthèse d'un superviseur.

1.3.2 Synthèse d'un superviseur

Dans le cadre de la théorie de supervision des SED, les spécifications peuvent être modélisées par un automate à état appelé automate de contraintes. Chaque état de l'automate correspond à plusieurs situations du procédé. Les transitions notées à partir d'un état, représentent les événements autorisés pour l'ensemble des états correspondants du procédé. Les autres événements (ceux non mentionnés) représentent les évolutions à éviter ou des évolutions physiquement impossibles.

Reprenons l'exemple de l'atelier de la figure 1.4, en lui associant la spécification K qui impose d'usiner les pièces d'abord sur la machine M_1 et ensuite sur la machine M_2 : La machine M_1 doit avoir accompli sa tâche β_1 avant que la machine M_2 ne commence la tâche α_2. L'absence de l'événement β_1 sur la transition, qui a comme état amont et état aval l'état 1, indique que le stock entre les deux machines est de capacité égale à 1 pièce. Les transitions notées à partir d'un état de l'automate de contrainte correspondent aux événements autorisés lors du fonctionnement en boucle fermée.

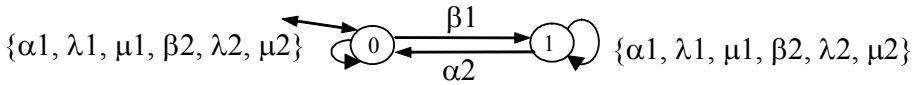


Figure 1.6. Automate modélisant la spécification à respecter

Ainsi, l'événement α_2 est interdit à partir de l'état 0 du graphe, ce qui permet d'exprimer le fait que la machine 1 doit avoir accompli sa tâche β_1 avant que la machine 2 ne commence la sienne.

Lorsque le langage décrit par la spécification est commandable et inclus dans le langage du procédé L(G), il peut être utilisé comme superviseur. Dans le cas contraire, il n'existe pas de superviseur S tel que L(S/G) = K. Par conséquent, il faut chercher une solution plus restrictive, i.e. le plus grand sous-ensemble de K qui soit unique et commandable supC(K) (figure 1.7), et qui respecte les restrictions imposées par K.

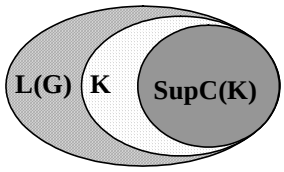


Figure 1.7. *Langage suprême commandable*

Un superviseur S sera dit optimal, si lorsqu'il est couplé à un procédé, on a l'égalité L(S/G) = supC(K). Cette propriété implique que si le langage de spécification K ne peut pas servir de superviseur, il est toujours possible de trouver un sous-ensemble de K qui soit commandable et qui respecte les spécifications.

Lorsque supC(K) se réduit à l'ensemble vide, alors c'est qu'il n'est pas possible de trouver un superviseur permettant de respecter les spécifications.

Il existe différents algorithmes permettant de vérifier la commandabilité d'un langage de spécification, et de générer le langage suprême correspondant [WON 87], [KUM 91].

1.3.3 Illustration

Pour illustrer les concepts de commandabilité et de supervision, nous prenons l'exemple de l'atelier de la figure 1.3 associé à la spécification de la figure 1.6. En appliquant l'algorithme de Kumar [KUM 91], nous obtenons le superviseur (figure 1.8) suivant :

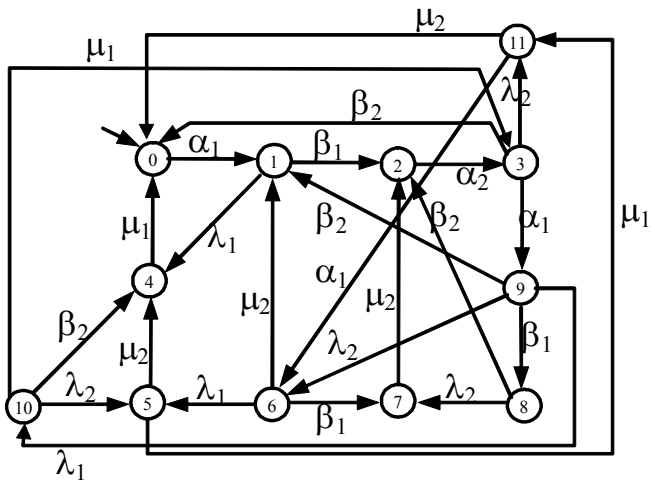


Figure 1.8. Automate du superviseur

Ce superviseur correspond à une stratégie simple. A partir de l'état initial, seul l'événement α_1, lié à la mise en marche de la machine 1, est autorisé. Si l'opération sur la machine 1 se termine normalement (état①), l'événement β_1 fait évoluer l'automate vers l'état ② où le superviseur autorise la mise en marche de la machine M_2 par l'intermédiaire de l'événement α_2. Par contre, si la machine M_1 tombe en panne, l'automate passe dans l'état ④ où seule la réparation μ_1 est autorisée, afin de revenir à l'état initial. Dès l'instant où la machine M_2 est mise en fonctionnement, le superviseur autorise la mise en service de la machine M_1 pour l'usinage d'une nouvelle pièce.

Un autre scénario d'évolution est par exemple de supposer que le superviseur autorise la mise en service de la machine M_1, à partir de l'état③, correspondant à la mise en marche de la machine M_2. Ainsi, l'événement α_1 fait évoluer l'automate vers l'état⑨ et si la machine M_2 tombe en panne, l'automate passe dans l'état ⑥ où trois évolutions sont possibles :

- La machine M_1 tombe en panne (état ⑤) et dans ce cas seule la réparation de la machine M_2 est possible(μ_2 est autorisée)
- L'opération sur la machine M_1 se termine normalement (état⑦) et il faut attendre de réparer la machine M_2 (état ②) pour la mettre en marche
- La machine 2 est réparée avant que l'opération sur la machine 1 se termine (état ①) et ainsi de suite.

2. DE LA SYNTHESE DE COMMANDE A L'IMPLANTATION

Dans le cadre de la théorie de supervision, le générateur est supposé produire des événements de façon spontanée, la seule manière pour que le superviseur affecte le comportement du système est alors d'autoriser ou d'inhiber des événements contrôlables. Ce superviseur résultant par l'un des algorithmes ([WON 87], [KUM 91]) peut être décrit par des automates où plusieurs événements contrôlables sont permis dans un état donné.

Il est à noter que la théorie de supervision ne prend pas en compte complètement l'aspect commande. En effet, un système de commande ne se contente pas que de l'autorisation et de l'inhibition d'événements, il doit en plus pouvoir forcer certains d'entre eux. Ce système de commande est souvent basé sur un ensemble de règles d'évolution prédéterminées proposant les actions appropriées. Ces actions correspondent aux événements contrôlables à appliquer sur le processus selon des observations, c'est-à-dire des événements incontrôlables provenant de la partie opérative. Après la synthèse de ces deux interprétations de ce qu'on appelle superviseur [RAM 89] et contrôleur [MAR 98], leur implantation exige des outils méthodologiques additionnels pour les rendre conformes à une architecture industrielle.

Pour cela, de nombreux travaux ont été réalisés pour modéliser le comportement d'un SED et concevoir sa commande par supervision ([CHA 99], [BAL 93], [BRA 96], etc…). Cependant, malgré la diversité de ces travaux, peu d'entre eux vont jusqu'à l'implantation de la commande élaborée.

Souvent l'utilisation des automates à états finis est adoptée pour spécifier la commande, et les automates programmables industriel (API) pour l'implanter. La spécification à base d'automates à états finis est une conséquence de l'utilisation de la théorie de supervision. L'implantation dans des API est liée à la prépondérance de cet outil pour la commande des processus industriels. Il faut préciser que les avantages ou les inconvénients de ces approches dépendent des outils utilisés et des démarches d'implantation préconisées.

2.1. Modélisation des systèmes à événements discrets pour la synthèse de commande

La mise en œuvre d'un algorithme de synthèse nécessite une modélisation détaillée de la partie opérative et des contraintes de sûreté et de vivacité. Les dynamiques du processus

et les contraintes limitent les évolutions possibles de la commande influencent le comportement de cette commande. Cependant, l'automatisation des systèmes complexes se fonde souvent sur un raffinement progressif des modèles [BRA 00] qui ne sont pas considérés dans le processus au préalable. En effet, modéliser des systèmes de complexité industrielle exige une analyse préliminaire, basée sur l'utilisation d'une abstraction plus ou moins formelle, sous une forme fonctionnelle, ensembliste,…, pour que le fonctionnement global d'un système puisse être conçu ou rénové ([MAR 98]).

Les méthodes classiques et les formalismes adoptés, pour modéliser un système industriel, à partir desquels les techniques de synthèse permettent la génération d'un superviseur montrent que :

- les modèles de procédé, sont des représentations détaillées, exhaustives, bien souvent «à plat» d'un système et souvent peu structurantes. On peut citer : les automates à états finis [SAK 03], les RdP [HOL 97], le langage à prédicats [SAN 95] ou encore la logique temporelle [FUS 83]. L'obtention de ces modèles est souvent difficile sans une analyse préalable, notamment à cause de leur taille, du manque de convivialité et d'expressivité des formalismes utilisés ou du manque de méthode.

- la définition d'une méthodologie de travail, qui pourra aider le concepteur à transformer progressivement les exigences de l'expert et les opérations du procédé en spécifications formelles du comportement prévu et des dynamiques du processus, doit être plus conviviale et expressive pour faciliter les techniques de synthèse automatique.

Pour répondre au manque de convivialité des modèles formalisés avec des automates à états [SHA 95], d'autres formalismes de modélisation ont été utilisés, par exemple une modélisation à base de règles proposée par Chandra et al dans ([CHA 01], [CHA 02]), et des algorithmes spécifiques de synthèse de superviseurs ont également été développés. C'est le cas par exemple des réseaux de Petri ([FAB 94], [GOD 96]), où il est possible de synthétiser un superviseur à l'aide de la théorie des régions comme le font Ghaffari et al [GHA 01], [GHA 02].

Pour diminuer l'explosion combinatoire liée en partie à la complexité de la phase de modélisation et aux algorithmes utilisés pour synthétiser un superviseur, des travaux utilisent une stratégie de décomposition des différents modèles pour limiter le nombre

d'états à traiter, ce qui conduit à introduire l'aspect modulaire, décentralisé et hiérarchique dans la théorie de commande par supervision [CHA 00b].

La décomposition peut concerner le modèle du procédé ou seulement la spécification du comportement attendu, ou les deux parties (commande et procédé) [CHA 00b]. Une présentation plus détaillée de ces approches telles que l'approche modulaire, l'approche hiérarchique et décentralisée est détaillée par la suite.

- L'approche modulaire (figure1.9) permet d'associer plusieurs contrôleurs de même niveau à un même procédé ([WNG 98], [QUE 02]). Ces différents contrôleurs travaillent parallèlement et en concurrence, ce qui peut conduire à des blocages et des conflits [BOU 02], citons par exemple l'exemple du forçage d'un événement par un contrôleur et son interdiction en même temps par un autre.

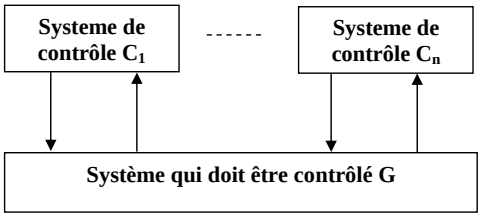


Figure 1.9. Contrôle par supervision modulaire [FEN 90]

- L'approche de supervision hiérarchique ([ZHO 90], [GOH 98]) se base sur l'utilisation des modèles simplifiés du procédé dans le but d'obtenir des superviseurs à un niveau d'abstraction plus élevé. Dans la figure 1.10, le modèle considère deux niveaux hiérarchiques dont le plus bas est constitué d'un procédé G_L et de son contrôleur C_L, et le niveau plus haut formé d'un modèle G_H et de son contrôleur C_H . Ces deux niveaux sont couplés. G_L est le procédé réel qui sera contrôlé par C_L alors que G_H est un modèle abstrait de G_L, utilisé avec son contrôleur C_H pour prendre les décisions plus globales et plus abstraites que les décisions prises au niveau bas.

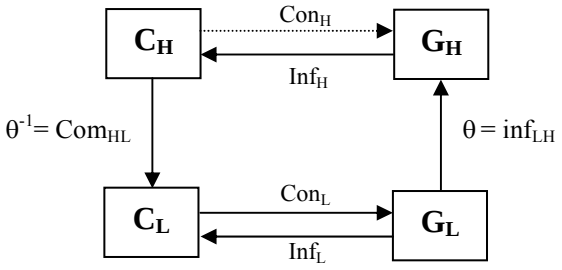


Figure 1.10. Structure hiérarchique

- L'approche par supervision décentralisée ([LIN 88], [LIN 90], [HAN.97], [YOO 02]) (figure 1.11) consiste à adopter une décomposition du système G en sous systèmes élémentaires ou procédés G_i. Chaque sous système possède son propre contrôleur local. Dans ce cas, les contrôleurs travaillent également en parallèle et en concurrence et peuvent créer des conflits, sauf en respectant des lois sur les modèles définis par [TAK 02]. Ainsi, un procédé G sera décomposé par des projections normales en divers sous ensembles G1 et G2 sur lesquels agiront des contrôleurs locaux [JIA 00].

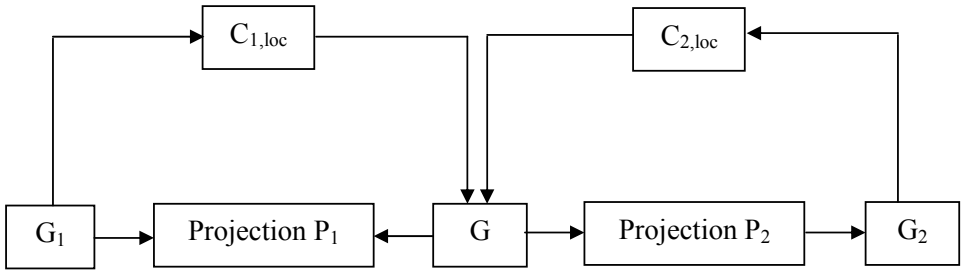


Figure 1.11. Contrôle supervisé décentralisé

L'avantage de ces approches est réel pour maîtriser la complexité de la phase de modélisation et donc le problème de l'explosion combinatoire. Cependant, les aspects méthodologiques en vue de l'implantation de la commande synthétisée ne sont pas complètement pris en compte.

2.2. Démarches d'implantation

La théorie de la supervision et les techniques de synthèse constituent un apport théorique important pour la conception des systèmes de commande à événements discrets, mais leur utilisation dans le milieu industriel reste très limitée.

On a vu précédemment que le procédé dans le cadre de la théorie de supervision produit des événements d'une manière spontanée et le superviseur affecte le comportement du processus en autorisant ou en inhibant des événements contrôlables. Ainsi, le superviseur est décrit par des automates où plusieurs événements contrôlables sont permis dans un état donné. Par contre, un système de commande réactif se doit de plus de pouvoir forcer l'occurrence de quelques événements. Ce constat conduit à séparer la notion de superviseur qui autorise et interdit de celle de contrôleur qui autorise, interdit et force les événements [CHA 99], [MAR 98].

Les travaux portant sur l'interprétation du superviseur et du contrôleur montrent des limitations de l'application de la théorie de supervision au milieu industriel. Ces limites sont dues aux méthodologies à appliquer, pour passer d'un superviseur ([RAM 87], [KUM 91]) à un contrôleur en vue de son implantation.

Nous présentons dans ce paragraphe différentes approches développées. Leur nombre et leur diversité témoignent de l'intérêt de la démarche de synthèse de commande, de son implantation et des difficultés de mise en œuvre. Chacune des approches diffère par les moyens et les méthodes utilisés en terme d'interprétation de la théorie de supervision adoptée, d'algorithme de synthèse utilisé et de démarche d'implantation.

La figure 1.12 résume les principales approches de synthèse de commande en vue de son implantation. Par la suite, une brève présentation de ces approches permet de mettre en évidence les problèmes, qui sont tant théoriques que pratiques, rencontrés lors de l'élaboration de la commande.

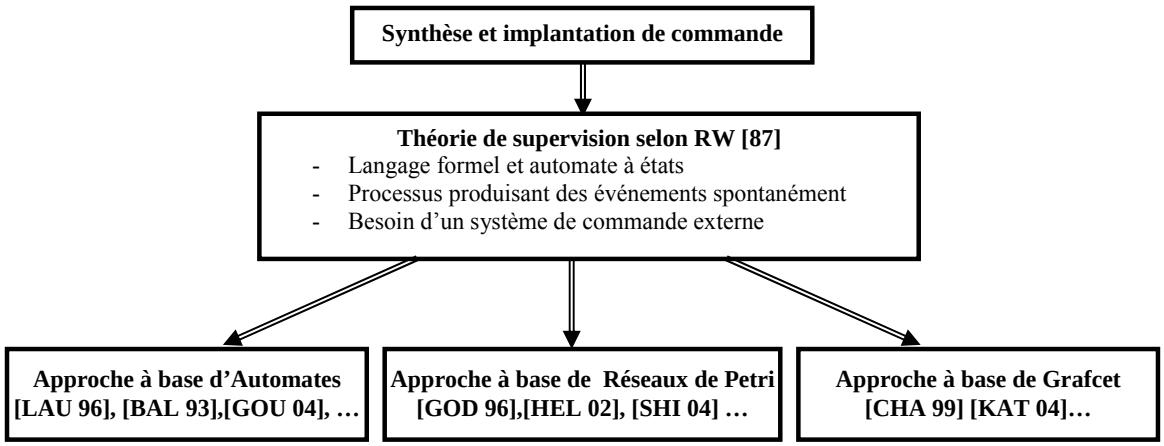


Figure 1.12. Vue structurée en fonction des formalismes des différentes approches de synthèse de commande

2.2.1 Approches à base d'automates

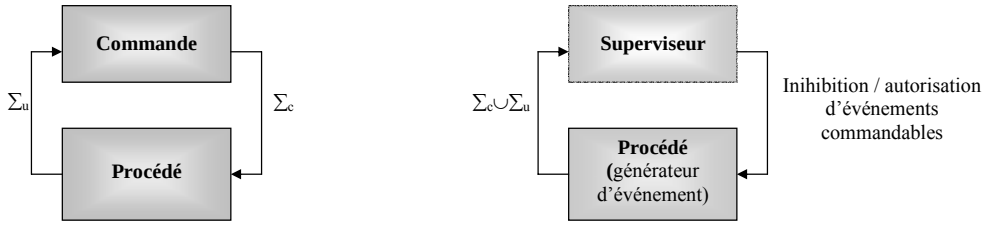
a. Approche de Balémi [BAL 93]

Synthèse de commande :

La démarche de Balemi consiste à modéliser le procédé, en se basant sur une interprétation des entrées-sorties (figure1.13.a) par rapport à la théorie de base (figure 1.13.b). Les entrées de la commande sont définies comme des événements

incontrôlables Σ_u générés par le procédé et les sorties de la commande sont définies par les événements contrôlables Σ_c envoyés vers le procédé.

Les contraintes de sécurité et de vivacité, ainsi que l'objectif de commande présentant les tâches à accomplir, sont décrits par des automates. Un superviseur représentant le comportement maximal et sans blocage du procédé vis-à-vis des contraintes ainsi qu'un contrôleur défini par le comportement maximal du procédé vis-à-vis des tâches à accomplir sont générés [RAM 89], et fonctionnent en parallèle durant l'exécution de la commande.



a. Interprétation en termes d'entrées/sorties b. Interprétation du modèle de base

Figure 1.13. Interprétation de la théorie

Implantation :

La figure 1.14 présente l'architecture d'implantation de la commande conçue par l'approche de Balemi. Cette architecture est structurée en trois blocs principaux : une unité de commande, une interface et les différents systèmes (Sys_i) physiques à commander. L'unité de commande désigne la commande globale. Celle-ci est composée d'un contrôleur et d'un superviseur. Le rôle du contrôleur est de garantir le respect des tâches à exécuter, tandis que le superviseur filtre les commandes provenant du contrôleur en garantissant le respect de toutes les contraintes de sécurité et de vivacité par autorisation ou interdiction des événements commandables à un état donné. Pour cela, le contrôleur génère des commandes relatives aux tâches à exécuter tenant uniquement compte du comportement possible du procédé à commander. Ces commandes sont considérées comme les entrées du système, auxquelles le procédé doit réagir, en générant des réponses. Par un processus de synchronisation, le superviseur inhibe les événements de commande qui vont à l'encontre des contraintes spécifiées. Par contre, toutes les réactions du système sont transmises au contrôleur sans restriction. Les commandes autorisées par le superviseur sont envoyées à l'interpréteur de commande, afin qu'elles soient exécutées. La transformation des informations logicielles en provenance du superviseur, en

informations physiques exécutables par les systèmes physiques est assurée par le module d'interprétation de commande/réactions, à travers diverses entrées/sorties.

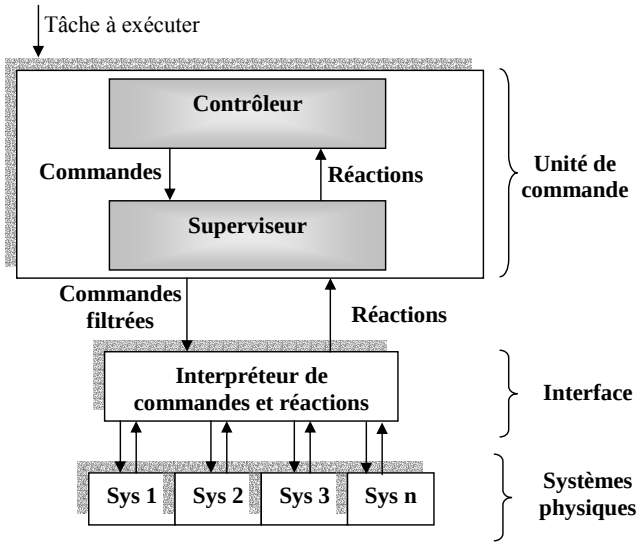


Figure 1.14. Architecture d'implantation d'une commande supervisée

Apport et limites :

Cette approche apporte une interprétation de la théorie de supervision qui soit adaptée à la commande des systèmes physiques réels. De plus, la séparation entre ce que doivent faire les systèmes, c'est-à-dire les tâches à exécuter et les contraintes à respecter, simplifie la tâche du concepteur relativement à la spécification des modèles. Toutefois, cette approche peut générer une explosion combinatoire exponentielle en nombre d'états car elle est centralisée et basée sur des automates à états.

b. Approche de Lauzon [LAU 96]

Synthèse de commande :

Dans la démarche de synthèse proposée, le procédé n'est plus considéré comme un générateur spontané d'événements tel que défini par la théorie de supervision [RAM 89], mais plutôt comme un système réagissant à des requêtes externes.

De façon plus explicite, durant la modélisation du procédé, tout événement commandable α est décomposé en un événement α_r correspondant à la requête externe, et un événement α_c correspondant à la prise en compte de la requête par le procédé.

La figure 1.15b présente l'application de cette notion à l'exemple figure 1.15a. Après cette interprétation, le superviseur est synthétisé selon la démarche préconisée par RW.

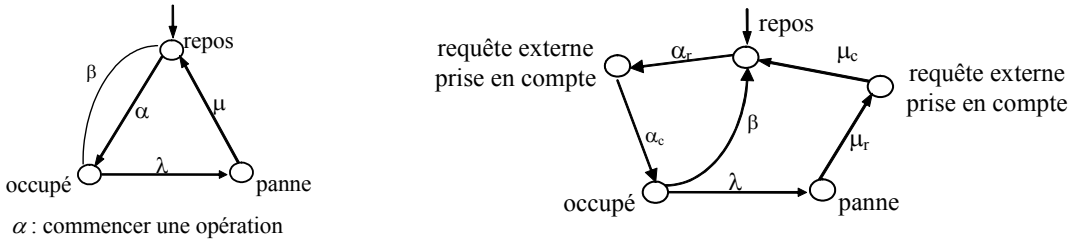


Figure 1.15. Modèle d'une machine simple

Le changement d'état dans le superviseur synthétisé après cette interprétation peut être donc effectué seulement quand la réponse (requête a été prise en compte) est reçue du procédé. Ainsi, le traducteur en langage API permettrait de définir le prochain état dans le superviseur en fonction de la requête demandée par le programme API. L'architecture d'implantation adoptée dans cette approche est détaillée par la suite.

Implantation :

L'architecture appliquée dans cette approche pour implanter la commande synthétisée au procédé, est représentée par la figure 1.16. L'utilisation conjointe d'un ordinateur de type PC et d'un automate programmable industriel API est adoptée. Sur l'ordinateur, se trouvent un générateur en ligne de commande ainsi qu'un traducteur automatique de la commande élaborée en langage à relais.

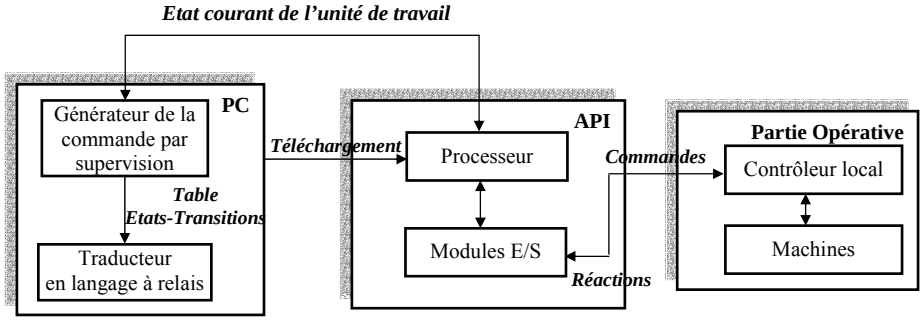


Figure 1.16. Méthodologie générale d'implantation

Dans la première phase, le générateur en ligne calcule la commande à appliquer de deux façons différentes : premièrement en calculant pas à pas la commande et

deuxièmement en développant une partie de la commande à implanter. Cette génération est développée en fonction des informations provenant de l'automate programmable puis transmise à un traducteur, qui de manière automatique génère le code relais correspondant à la table. En dernière phase, ce code est chargé sur un automate programmable industriel en vue de son exécution. Pour chaque unité de travail de la partie opérative, on dispose d'un contrôleur local, chargé du suivi des machines. Suite à l'accomplissement d'une tâche prédéfinie ou à l'occurrence d'un événement non prévu dans la commande courante, une nouvelle commande basée sur le modèle de base du procédé et des contraintes est calculée, traduite et chargée sur l'automate.

Apport et limitation :

L'intérêt de cette architecture d'implantation est de permettre la synthèse en ligne de la commande sans que l'on s'occupe de l'encombrement de la mémoire nécessaire lors du pilotage des systèmes de grande taille. Par ailleurs, cette approche permet la traduction automatique de la commande pour implantation. Cependant, la modélisation explicite de la partie opérative est une tâche complexe qui se heurte non seulement à des problèmes méthodologiques de structuration et de composition de ses éléments mais aussi au choix de la granularité du comportement modélisé. En plus, le fait d'utiliser une modélisation intuitive de la partie opérative par des automates à états, peut restreindre la prise en compte de tout le comportement de la partie opérative en intégrant implicitement des contraintes de sécurité et peut aussi provoquer des erreurs de modélisation, ce qui peut conduire à des blocages lors de la synthèse de la commande.

c. Approche de Gouyon et *al* [GOU 02]

Synthèse de commande :

Cette démarche repose sur une méthode modulaire ([GOU 02], [GOU 04]) de synthèse, et plus précisément, sur la synthèse d'une commande hiérarchisée dont la structure se base sur la décomposition modulaire des processus physiques à commander et une analyse critique des superviseurs générés vis-à-vis des modèles de spécifications retenus.

Cette démarche permet de synthétiser une architecture hiérarchique coordonnée de superviseurs basée sur la structure du système physique commandé. Elle se base, pour cela, sur une mise en œuvre itérative des algorithmes de synthèse qui consiste à réutiliser

des superviseurs d'un niveau inférieur pour construire le modèle de procédé d'un superviseur de niveau supérieur, et ce, depuis les niveaux les plus technologiques.

La première étape de cette démarche consiste en la synthèse de superviseurs de bas niveau, ne contrôlant chacun qu'un seul élément de la partie opérative. Après la synthèse de superviseurs de niveau 1, la phase suivante est de synthétiser les superviseurs de coordination de différents éléments de niveau plus élevé de façon récurrente. Par exemple les superviseurs de niveau n+1 assurent la coordination de plusieurs superviseurs de niveau n. Dans un objectif de synthèse de ces superviseurs de coordination, l'approche se base sur une première étape de projection où les différents superviseurs du niveau n sont projetés pour ne conserver que les événements observables de l'alphabet de départ, une deuxième étape de changement de contrôlabilité où les événements en entrée (resp. sortie) qui étaient vus par le niveau n comme incontrôlables (resp. contrôlables) sont vus par le niveau n+1 de façon complémentaire et enfin une dernière étape de composition synchrone des projections des différents superviseurs de niveau n. Le modèle utilisé pour synthétiser le superviseur de niveau n+1 est composé par l'ensemble des modules de commande de niveau n, mais uniquement sur les parties de leurs alphabets correspondant à des événements de ce niveau n (Figure 1.17).

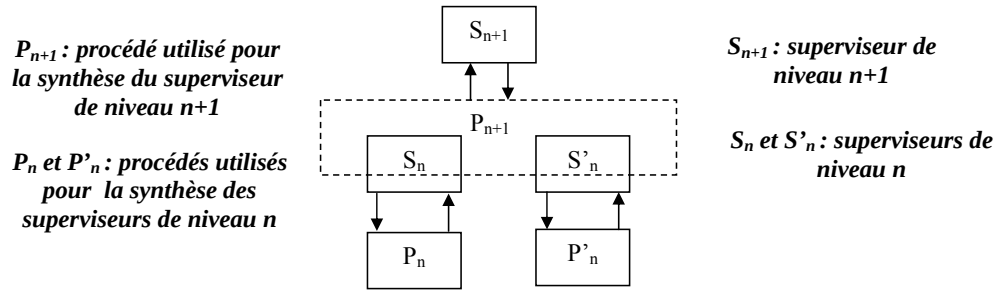


Figure 1.17. Procédé utilisé pour la synthèse d'un superviseur de niveau n+1

Implantation de la commande obtenue par cette démarche :

Cette démarche méthodologique conduit à la synthèse par récurrence de superviseurs à chaque niveau de la structure de commande du système automatisé. Cette démarche est résumée dans la figure 1.18.

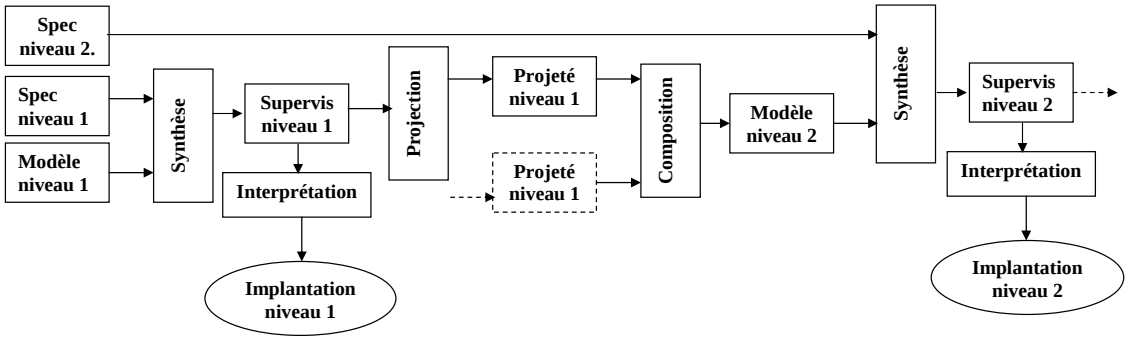


Figure 1.18. Démarche de synthèse de commande.

Les superviseurs synthétisés par cette démarche ne sont pas directement implantables car indéterministes et modélisés par des automates à états. De plus, ces superviseurs, qui interdisent et autorisent des événements, sont interprétés en des contrôleurs qui forcent des sorties d'un API.

Le principe proposé est alors d'effectuer une traduction directe du superviseur obtenu par synthèse dans un langage de programmation des API en appliquant des techniques d'interprétation synchrones [LHO 97]. Pour pallier l'indéterminisme de l'automate synthétisé, cette démarche définit des propriétés pour chaque groupe de transitions sortant d'un même état, en fonction de leur hiérarchie puis en fonction de leur contrôlabilité (figure 1.19). Autrement dit, les événements de niveaux hiérarchiques supérieurs sont prioritaires sur les événements de niveaux hiérarchiques inférieurs et les événements incontrôlables sont prioritaires sur les événements contrôlables.

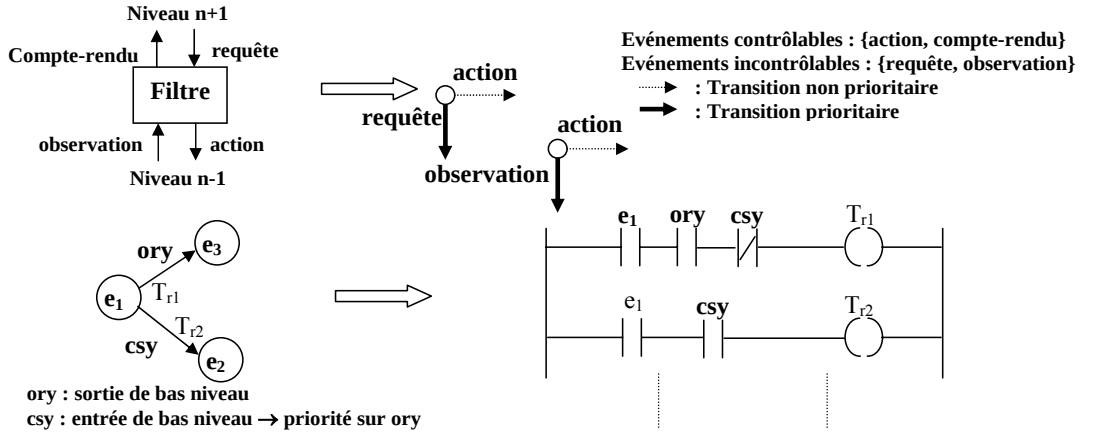


Figure 1.19. Interprétation pour implantation.

Apports et limites :

Cette méthodologie de synthèse et d'implantation montre d'une part l'intérêt d'une méthode, basée sur la structuration d'un système de commande, pour réaliser une synthèse

modulaire facilitant la modélisation du procédé à commander et l'implantation des superviseurs dans des langages de programmation. D'autre part, la modélisation des spécifications est facilitée, puisque les propriétés que doit respecter le procédé sont modélisées de manière parcellaire relativement à la mission associée à un superviseur donné. En particulier, la spécification relative à la synthèse d'un contrôleur de niveau n ne tiendra pas compte des contraintes de niveau n-2 qui sont, elles, prises en charge par le superviseur de niveau n-1. Néanmoins, l'écriture de ces spécifications ainsi que les sous modèles de partie opérative reste une tâche difficile ayant un impact sur la procédure de synthèse. En plus, le nombre de superviseurs à traduire pour implantation dans un API peut nuire à la praticabilité de cette approche.

d. Approche de De Queiroz et al [QUE 02]

Synthèse de commande :

Cette démarche propose une approche modulaire locale [QUE 00] représentant une extension des résultats de RW pour réduire l'explosion en espace d'état dans un système composé. Cette approche réduit la complexité du processus de synthèse ainsi que la taille des superviseurs en exploitant la modularité de la partie opérative et les spécifications. Au lieu d'un superviseur global pour un procédé global, cette démarche permet d'obtenir pour chaque spécification, un superviseur modulaire pour chaque sous système. La modularité est vérifiée par une condition nécessaire et suffisante nommée la modularité locale. Cette condition assure que l'ensemble des superviseurs locaux possède les performances du superviseur global.

Implémentation de la commande synthétisée :

La démarche de synthèse adoptée fournit un ensemble de superviseurs modulaires locaux réduits, représentés par des machines à états finis (figure 1.20). Dans chaque état s_i des superviseurs, la commande est définie comme une interdiction d'un ensemble d'événements α_i précédemment calculés. Puis, l'implémentation physique du système de commande consiste à exécuter les automates des superviseurs en parallèle, selon les événements lus du système réel, et à envoyer les signaux d'interdiction au procédé, selon l'état actuel du procédé.

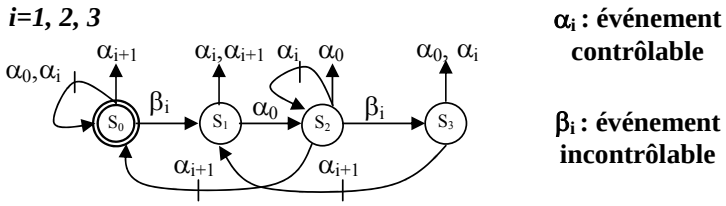


Figure 1.20. Automates superviseurs réduits

Visant à exécuter les superviseurs modulaires, le système de commande est programmé dans une hiérarchie de trois niveaux, comme illustré par la figure 1.21.

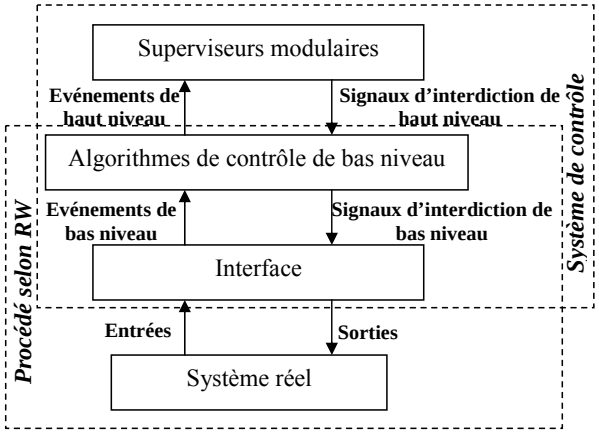


Figure 1.21. Structure du système de contrôle

Le premier niveau est le niveau le plus élevé concernant l'ensemble des superviseurs modulaires locaux réduits, conçus en se basant sur la théorie de supervision selon RW. Ces superviseurs ne contiennent pas nécessairement l'information complète sur le comportement du procédé. Par conséquent, les modèles machines d'état correspondants aux sous systèmes sont implémentées concurremment au deuxième niveau contenant les algorithmes de contrôle de bas niveau. Ces algorithmes implantés dans ce bas niveau de commande envoient des événements de haut niveau correspondant aux signaux d'interdiction permettant de mettre à jour les superviseurs modulaires, dès que les signaux sont traités par le procédé réel. L'évolution parallèle des sous systèmes asynchrones suit les signaux d'interdiction de bas niveau (transitions contrôlables) et les événements de bas niveau (transitions incontrôlables) provenant du niveau de l'interface, signalant des changements d'état aux superviseurs.

Le système de contrôle est implanté dans un API en langage LD [IEC 93]. Les machines d'états sont programmées en LD comme proposé par [FAB 98]. Les états et les signaux internes du système de contrôle sont présentés par des « flags ». Dans le haut

niveau, les superviseurs sont implémentés concurremment avec l'ensemble des signaux interdits correspondants aux états actifs (figure 1.22a). Au niveau des algorithmes de contrôle, les machines d'états sont programmées de manière asynchrone. Les événements commandables sont programmés de telle façon qu'ils seront activés s'ils ne sont pas interdits par les superviseurs (figure 1.22b). L'interface est programmée comme des machines à états parallèles (figure 1.22c). La mise en marche (c-start) de cette interface est effectuée par les algorithmes de contrôle de bas niveau. Cette interface de bas niveau, produit des signaux de sortie (O0.1) du système de commande, lit les signaux d'entrée (I 0.1), et fournit les algorithmes de contrôle par des réponses logiques (c-end) qui reflètent l'occurrence des événements incontrôlables produits par le système réel.

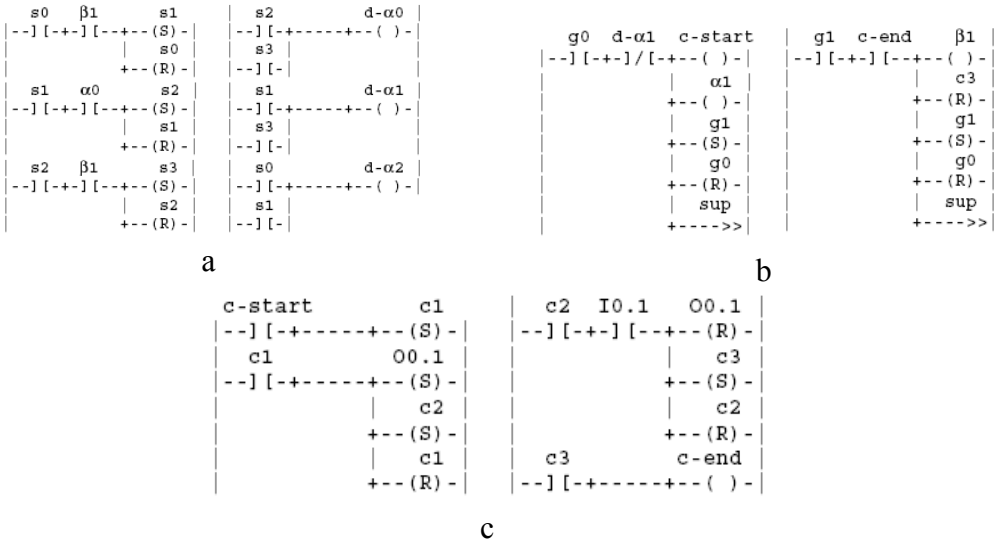


Figure 1.22. Implantation en ladder des différents niveaux du système de contrôle

Apports et limites :

Cette approche propose une méthodologie pour la synthèse de commande supervisée qui a été appliquée avec succès à une cellule de production commandée par un API. L'approche modulaire locale, alliée à un algorithme de réduction du superviseur, a été utilisée pour la synthèse des superviseurs réduits et optimaux.

Cette approche propose aussi une structure générique pour l'implantation d'un système de commande. Cette structure utilise les superviseurs parallèlement, pour exécuter le programme et se connecter par une interface au système réel d'une façon intelligente. Cependant, la vérification de la condition de la modularité locale peut nuire à l'applicabilité de cette approche sur des systèmes plus complexes, sachant que cette

condition est vérifiée en effectuant une composition de tous les superviseurs synthétisés [WON 88] [QUE 00]. En plus, l'adoption d'une modélisation non structurée à base d'automates peut conduire à des erreurs de modélisation, donc à des blocages en exécution.

2.2.2 Approches à base de réseaux de Petri

Les approches présentées auparavant utilisent des automates à état pour générer la commande et fournissent un cadre pour établir des propriétés fondamentales pour le problème de synthèse de commande des systèmes à événements discrets. Cependant, ces approches ne présentent pas de modèles structurés pour des systèmes réels et sont caractérisées par le nombre d'états qui croient exponentiellement en nombre d'états pour des systèmes plus complexes, ce qui limite les possibilités de développer des algorithmes efficaces pour l'analyse et la synthèse de la commande.

Ces limitations ont poussé à l'utilisation des modèles plus évolués tels les RdP pour fournir des descriptions plus compactes dans le contexte de la théorie de supervision [GIU 96], car la structure du réseau peut être maintenue petite dans la taille même si le marquage augmente.

Plusieurs techniques de synthèse se sont développées par la suite pour adapter les premiers résultats au contexte des réseaux de Petri. [GUI 92] [HOL 90], [LI 93], [MOO 94], [YAM 96], proposent chacun une solution pour la synthèse de commande par RdP. Cependant, dans chaque cas, des hypothèses restrictives limitent l'applicabilité de l'approche.

S'il existe de nombreux travaux liés à la synthèse d'une commande supervisée à partir de RdP, peu de ces travaux se sont intéressés à l'implantation de cette commande. Parmi ces travaux, on trouve ceux de Zhou et Dicesare ([ZHO 93]) qui consistent à obtenir une commande implantée dans un PC et directement liée aux équipements du système réel. D'autres approches sont basées sur une transformation du RdP, soit en un programme Ladder ([PEN 04] [TZA 02] [PAR 00]), soit en SFC [HEL 01], exécutable dans un API ([JAL 94], [ZHO 98]).

Nous allons présenter ici deux approches qui consistent à implanter la commande spécifiée par RdP. La première approche est celle proposée par Hellgren et al ([HEL 01], [HEL 02]). Celle ci consiste à générer la commande utilisant le RdP pour modéliser le

procédé et les spécifications puis à l'implanter dans un langage SFC de la norme IEC 1131-3. Pour diminuer l'explosion combinatoire, la spécification du procédé et des contraintes se fait de façon modulaire pour ensuite extraire les superviseurs locaux, selon la théorie de supervision [RAM 89]. Pour que l'implantation de ces superviseurs soit compatible avec le choix du langage SFC, les problèmes de déterminisme et de synchronisation doivent être pris en compte.

Pour éviter ces problèmes, des techniques et des hypothèses ont été mises en place en utilisant des propriétés sur les transitions et une méthode de « paramétrisation » [HEL 02]. En plus, la démarche étant modulaire, une méthode de synchronisation a été développée. Dans cette approche, l'implantation en SFC (figure 1.23) se base sur une interprétation du RdP tout en passant par quelques étapes :

① Une méthode appelée « paramétrisation » est utilisée pour obtenir des modèles déterministes pour l'implantation dans un API [HEL 02].

② Une conversion des modèles superviseurs du RdP en langage SFC est réalisée.

③ Une synchronisation des différents modèles prend en compte les interactions entre les différentes ressources du procédé d'une part, et les différentes spécifications d'autre part.

④ Une résolution du problème de causalité [FAB 98], détermine la source des événements générés. Pour cela, une construction d'un module appelé générateur d'événements (*Monitor event*) [HEL 01] est utilisée.

Cette approche permet alors d'effectuer une interprétation selon certains critères liés à l'implantation en SFC. La difficulté d'une telle approche, réside alors dans la vérification des hypothèses liées au déterminisme et à la synchronisation. Le choix d'une implantation modulaire est donc plus difficile qu'une démarche globale.

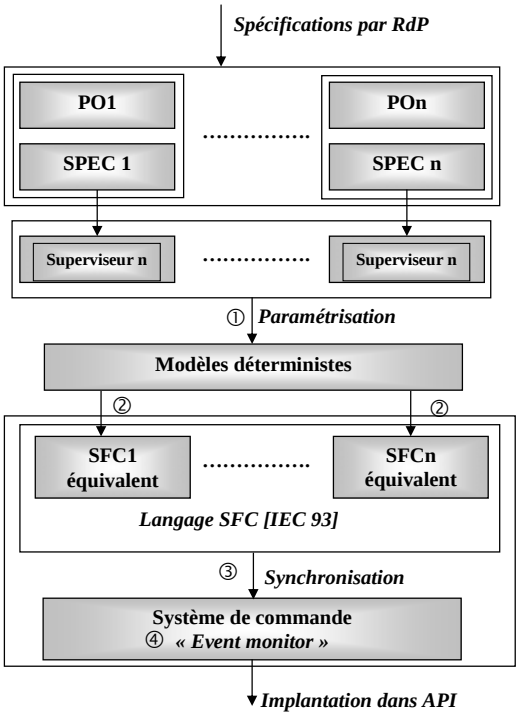


Figure 1.23. Architecture d'implantation dans un API.

La deuxième approche étudiée est présentée par Godon [GOD 96], elle consiste à spécifier le comportement libre du procédé par des réseaux de Petri synchronisés. Les contraintes à respecter sont décrites par une liste de marquages interdits. Un modèle RdP décrivant le plus large comportement qui respecte les contraintes, est ensuite élaboré. L'application du formalisme lié aux RdP permet de garantir l'absence de blocage, afin d'obtenir un superviseur au sens de RW, qui est implanté en tant que modèle de commande.

L'architecture d'implantation est représentée dans la figure 1.24. L'ensemble réside dans un PC, lié au procédé soit par le biais d'une carte d'E/S classique, soit par le biais d'une carte microcontrôleur. Dans cette approche, en plus du risque d'explosion combinatoire dû à l'utilisation d'une approche centralisée, le fait d'implanter le superviseur en tant que modèle de commande suppose que les objectifs de commande aient été intégrés dans la description du système. Cette démarche complexifie la tâche de modélisation pour le concepteur.

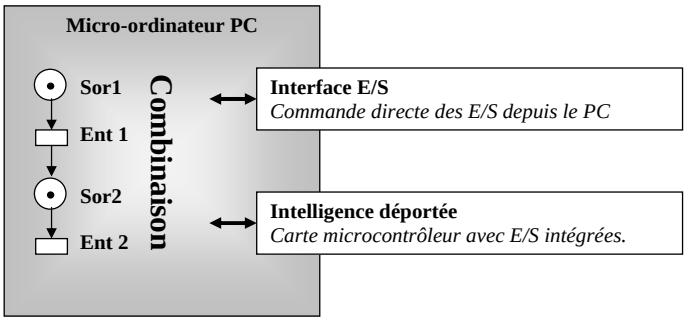


Figure 1.24. Schéma de commande utilisant deux types d'interface

2.2.3 Approches à base de Grafcet

La commande d'un système à événement discret est considérée comme un système réactif en interaction avec une partie opérative. La modélisation de cette commande prend en compte l'environnement du système commandé, c'est-à-dire que le système réagit aux évolutions de ses entrées et génère des sorties qui vont modifier l'état de la partie opérative. Si le temps de réaction de la commande est plus court que celui de la partie opérative, une hypothèse doit être ajoutée en considérant les entrées et les sorties qui en résultent comme synchrones [LES 98]. Pour cette raison, l'utilisation d'un modèle du procédé, sous forme d'automates à états par exemple, comme étant un modèle de commande opérationnelle est à éviter. Le modèle qui répond à cette exigence, est le Grafcet ou un langage API.

Il existe peu de travaux ([NDJ 99], [CHA 99], [KAT 04],) liés à la synthèse d'une commande basée sur le grafcet. Nous présentons ici une de ces approches proposées au Laboratoire d' Automatique de Grenoble par Charbonnier [CHA 96] et reprise ensuite par Kattan [KAT 04]. La figure 1.25 représente cette démarche de synthèse et d'implantation d'une commande supervisée par grafcet proposée dans [CHA 96].

L'approche de Charbonnier considère le procédé comme étant une composition entre un procédé à commander et un système de commande. Le procédé étendu alors obtenu est un générateur d'événements en interaction avec un superviseur, conformément à la théorie de supervision. La dynamique de la commande et du superviseur est modélisée par des Grafcets. Pour que la sémantique du superviseur soit respectée, les Grafcets ne manipulent, en entrées et en sorties, que des événements.

1- La commande est spécifiée par un ensemble de grafcets partiels décrivant les tâches à exécuter. Ces grafcets incluent les comportements supposés du procédé. Ils

représentent ainsi le modèle dit le procédé étendu.

2- Les contraintes de sécurité et de vivacité de l'application sont spécifiées par des grafcets appelés « de supervision ». Ceux-ci servent à figer les actions des grafcets de commande.

3- Chaque grafcet partiel de commande est traduit, suite à l'introduction du figeage, en automate événementiel à états finis. Une traduction similaire est élaborée pour les grafcets de supervision.

4- La composition des différents automates résultant de la troisième étape permet d'obtenir le modèle du fonctionnement désiré.

Lors que le système devient commandable, l'implantation de la commande se fait en implantant les grafcets de commande et ceux de supervision sur un API.

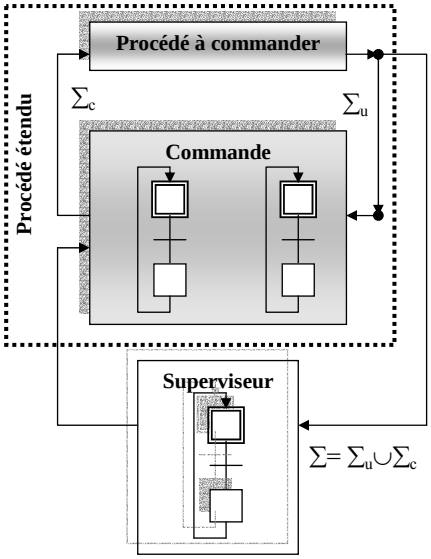


Figure 1.25. Schéma global de la démarche

Kattan [KAT 04] a repris le travail de Charbonnier [CHA 96] en développant deux extensions pour la synthèse de contrôleur des systèmes à événements discrets modélisés par Grafcets. La première contribution consiste à synthétiser un automate de superviseur dans le cas où le langage des spécifications n'est pas contrôlable par rapport au langage du procédé étendu. Pour implanter l'automate de superviseur obtenu, une méthode systématique de passage de l'automate superviseur au Grafcet superviseur de manière structurelle, a été élaborée.

Toutefois, dans cette approche, le Grafcet final de superviseur obtenu contient un nombre important d'étapes, identiques au nombre d'états de l'automate synthétisé. La complexité des superviseurs reste donc constamment conséquente. C'est par cette raison que pour Kattan a développé une approche de synthèse structurelle, dans laquelle la taille du Grafcet obtenu est réduite et implantable dans des automates programmables. Cette méthode est basée sur les invariants de marquage qui permettent de déterminer un certain nombre d'étapes (figure1.26), appelées étapes de contrôle, à ajouter au modèle initial pour faire respecter les spécifications de commande.

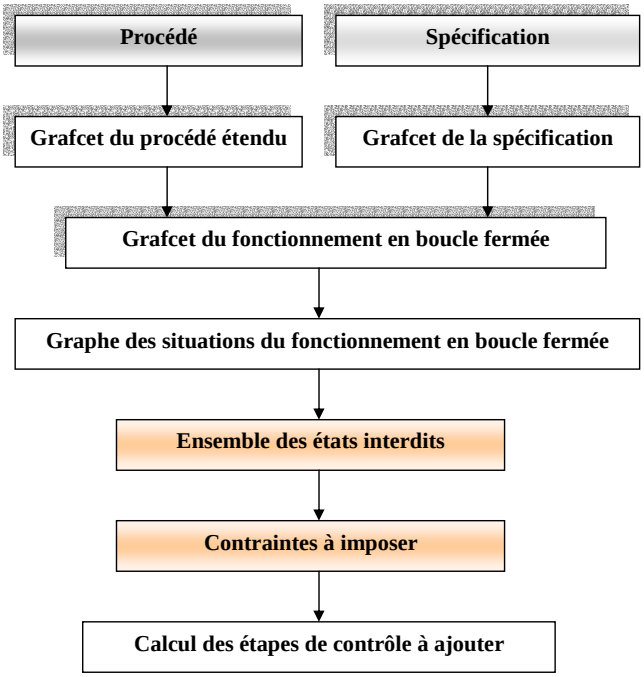


Figure1.26 Les étapes nécessaires pour synthétiser le contrôleur.

Apports et limitations

L'approche [CHA 96] précédente a présenté les travaux relatifs à l'association du Grafcet et de la théorie de supervision. Elle a donné lieu à son implantation dans l'atelier inter-établissement de productique de Dauphiné Savoie (Grenoble) à base de 77 Grafcets et 200 entrées/sorties. L'utilisation des Grafcets séquentiels avec figeage pour décrire les contraintes de supervision, constitue une commodité intéressante de modélisation par rapport à une description par automates. Malheureusement, il n'est pas possible de retrouver la structure de Grafcet du départ car la taille du Grafcet obtenu correspond à celle de l'automate du superviseur optimal. Ce résultat limite l'application de cette technique à des systèmes plus complexes et a conduit [KAT 04] à proposer une méthode

de synthèse de commande structurée basée sur le Grafcet. Toutefois, ces deux approches sont limitées à une sous classe de Grafcets dont les réceptivités sont décrites uniquement par des événements simples. De plus, elle ne préconise pas une modélisation explicite du procédé, ce qui rend difficile l'expression des contraintes induites par la partie opérative, et ne garantit pas le non blocage, lors de l'exécution réelle de la commande.

Dans le cadre des travaux de recherche de notre équipe, une démarche a été développée ayant pour objectif de synthétiser pour un procédé donné, une implantation de la commande qui soit la plus large possible par rapport à une commande spécifiée par Grafcet et un ensemble de contraintes de sécurité et de vivacité représentant le comportement autorisé. Nous allons présenter complètement dans le chapitre 2 cette démarche en l'illustrant sur une application pour mettre en évidence ses apports et faire ressortir les difficultés rencontrées lors de son élaboration.

3. CONCLUSION

Dans ce chapitre, nous avons présenté la théorie de supervision des SED. Cette théorie, initiée par Ramadge et Wonham, est basée sur des concepts formels tels que les langages formels et les automates pour la modélisation et la synthèse de la commande des SED. L'objectif de la théorie de supervision est la synthèse d'une entité externe au système, appelée superviseur, capable de faire respecter à la commande un comportement désiré en tenant compte des possibilités technologiques, des contraintes fonctionnelles, des contraintes de sécurité etc. Le superviseur agit par inhibition d'événements contrôlables et doit tolérer tous ceux qui sont incontrôlables.

Cependant, cette théorie considère d'une part que, le procédé génère des événements spontanément sans système de commande externe pour forcer des événements. D'autre part, elle manque de convivialité en raison de l'utilisation de langages formels ou de graphes d'états rudimentaires pour spécifier la commande, ce qui conduit à l'explosion combinatoire lors de la synthèse d'un superviseur. Si les premiers résultats de cette théorie doivent permettre la synthèse de la commande sur des cas d'étude de taille réduite, leur application pour la synthèse de commande au niveau industriel pose toujours un problème.

Plusieurs travaux se sont développés pour résoudre ces difficultés. Mais, très peu concernent la manière d'implanter une commande issue de la théorie de supervision car les procédures de synthèse fournissent un modèle d'un superviseur abstrait qui ne peut être implanté en l'état dans un système réel. Ces approches privilégient majoritairement le modèle automate pour spécifier la commande et l'Automate Programmable Industriel comme environnement d'implantation.

Les approches à base d'automates, de réseaux de Petri et à base de Grafcet que nous avons présentées dans ce chapitre proposent chacune une méthodologie de synthèse de commande en vue de son implantation dans un API. Cependant, dans chaque approche, des hypothèses restrictives limitent l'applicabilité.

La figure 1.27 illustre les principales améliorations introduites par certaines approches, vis-à-vis des objectifs attendus de la synthèse de la commande supervisée.

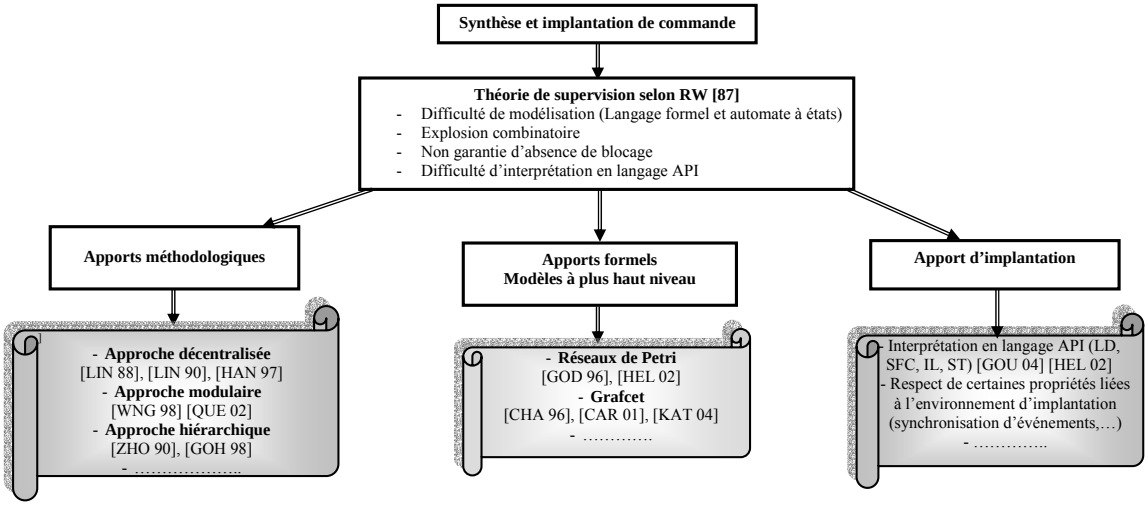


Figure 1.27. Les trois axes d'amélioration de la synthèse de commande supervisée

Chapitre 2

SYNTHESE AUTOMATIQUE DE COMMANDE SPÉCIFIÉE PAR GRAFCET

Nous présentons dans ce chapitre le travail mené au laboratoire autour de la synthèse de commande spécifiée par Grafcet, basée sur la théorie de supervision des SED. La première partie du chapitre est consacrée à l'étude comportementale du Grafcet qui a montré la nécessité de prendre en compte la partie opérative pour identifier correctement le comportement d'un système réel, lors de son exécution. Nous présentons ensuite la démarche de synthèse de la commande, à travers les étapes nécessaires pour passer d'une spécification exprimée en Grafcet à son implantation dans une architecture programmable. Nous abordons ensuite nos travaux proprement dits, basés sur cette démarche de synthèse de commande.

1 INTRODUCTION

L'origine du Grafcet est liée aux travaux de la commission de normalisation de l'AFCET en 1977. Cette commission animée par Michel Blanchard [BLA 79] avait pour objectif de définir un outil de modélisation des cahiers des charges des automatismes industriels. Ces travaux ont ensuite été repris et ont donné lieu en 1982 à une première normalisation [AFN 82]. En 1988, le Grafcet fait l'objet d'une norme internationale [IEC 88], qui remplace la précédente et définit l'établissement des diagrammes fonctionnels pour les systèmes de commande. Depuis, des modifications de cette norme sont apparues en 1991 [IEC91] et en 2002 [IEC 02].

L'utilisation du Grafcet est particulièrement adaptée pour spécifier et modéliser la commande des systèmes séquentiels. Cet outil graphique permet de représenter simplement les entrées-sorties du système avec leurs liens, les contraintes de parallélisme, de synchronisation et de séquencement. L'utilisation de ce modèle dans la spécification de la commande a conduit à définir la norme SFC (Sequential Function Charts) [IEC 93] pour l'implantation de la commande dans un Automate Programmable Industriel.

Les caractéristiques définies ci-dessus procurent au Grafcet une sémantique très riche. Cette richesse rend plus complexe la description exhaustive du comportement du Grafcet [LHO 97], mais une telle description est nécessaire afin de procéder à la synthèse d'une commande correcte. Cette opération de synthèse contribue à améliorer la sûreté de fonctionnement des systèmes automatisés puisqu'elle permet de renforcer la confiance accordée, par les utilisateurs et les concepteurs, dans les modèles établis. En effet, la synthèse permet de disposer d'une implantation correcte de la commande à partir d'une spécification exprimée en Grafcet et des contraintes à respecter sur les évolutions du procédé.

L'objectif de ce chapitre est de présenter les travaux effectués autour de la synthèse de commande spécifiée par Grafcet [CAR 02]. La section 2 présente les différentes étapes qui ont été identifiées pour passer d'une spécification Grafcet à l'implantation de

la commande correspondante. Pour formaliser ces étapes, le choix d'outils/méthodes ayant de solides bases formelles, tels que les automates et la théorie de supervision des SED a été fait et conduit à la démarche proposée en section 3. L'objectif de la section 4 est alors de démontrer l'applicabilité de cette démarche formelle sur un exemple réel et d'énumérer les difficultés qui constituent un frein à l'utilisation de la démarche en milieu industriel.

2 SYNTHESE DE COMMANDE SPECIFIEE PAR GRAFCET

Les travaux effectués dans notre équipe concernant la synthèse d'une commande spécifiée par Grafcet sont basés sur une étape fondamentale qui consiste à obtenir un modèle comportemental du Grafcet. A partir de cette connaissance du modèle comportemental, nous avons pu définir les étapes nécessaires pour atteindre l'objectif d'implantation d'une commande sûre, déterministe, réactive et sans blocage.

Le modèle recherché devra être déterministe et réactif. Le déterminisme se traduit par une commande qui génère des sorties identiques pour un même scénario d'évolution des entrées et la réactivité se traduit par la faculté de la commande à réagir immédiatement aux évolutions des entrées. Par conséquent, pour assurer le déterminisme et la réactivité, le modèle de la commande doit évoluer instantanément d'une situation stable à une autre suite à l'occurrence d'un événement de l'environnement.

2.1 Etude Comportementale du Grafcet

Pour étudier le comportement dynamique du Grafcet, le graphe d'états est choisi comme élément de base de la représentation des systèmes séquentiels car il permet de décrire les différents états possibles du système ainsi que les transitions entre ces états. Plusieurs approches [BLA 79], [ROU 94], [GAF 96] s'appuient également sur l'utilisation de ce graphe pour identifier le comportement du Grafcet.

L'objectif de l'étude du comportement du Grafcet est de faire ressortir, de manière graduelle, les éléments essentiels à prendre en compte pour identifier le comportement du Grafcet. Tout d'abord, il faut identifier le comportement du Grafcet indépendamment de son environnement et de la partie opérative qu'il commande. Cette étude a permis de recenser quatre étapes construisant par raffinements successifs le graphe d'états [ZAY 99]. On construit d'abord le graphe d'état à partir de la structure du Grafcet, puis des équations logiques des réceptivités, ensuite la notion de stabilité est prise en compte, enfin ce sont les postulats temporels du modèle Grafcet qui sont considérés (figure 2.1). Chacun des graphes d'états successifs décrit un comportement particulier du Grafcet, mais il existe un écart entre ces différents comportements et le comportement effectif durant l'exécution réelle. Pour combler cet écart et de manière à

identifier correctement le comportement du Grafcet lors de l'exécution réelle, l'introduction d'un cinquième élément qui est la partie opérative est nécessaire.

Un algorithme d'extraction du graphe d'état (automate) correspondant à une mise en œuvre formelle des étapes de raffinement du graphe d'états présentées ci-dessus est développé au LURPA [ROU 94] et implanté dans l'outil *AGGLAE*.

L'automate des situations stables (ASS*)* obtenu par *AGGLAE* permet d'évoluer d'une situation stable à une autre dès l'occurrence d'un événement de l'environnement et entraîne l'activation et la désactivation des actions relatives à la dite situation. Il correspond donc à un modèle de commande réactif et déterministe. Les états de l'automate ASS ainsi générés représentent les situations stables du Grafcet. Chaque état est donc caractérisé par l'ensemble des actions à activer et l'ensemble des étapes actives du Grafcet lors de la situation correspondante. Les transitions de ASS correspondent aux évolutions du Grafcet d'une situation stable à une autre. L'expression logique associée à ces transitions, est composée des entrées du Grafcet et de fronts sur ces entrées.

L'automate ASS contient généralement de nombreuses instabilités liées à la nature exhaustive des hypothèses faites sur la valeur des variables d'entrées. Ces mêmes hypothèses introduisent aussi des situations de commande qui ne seront jamais exécutées, remettant ainsi en cause l'exactitude du Grafcet. Pour que cet automate reflète le comportement effectif de la commande au sein du système global il faut intégrer la partie opérative. Ainsi, la section 2.2 a pour objet de montrer la nécessité mais aussi la difficulté de la prise en compte de la partie opérative.

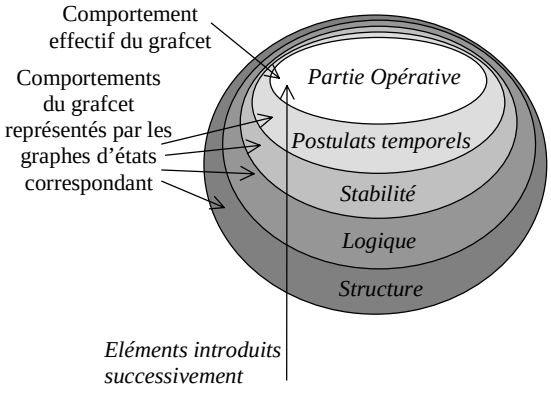


Figure 2.1. Différents éléments pour définir le comportement d'un Grafcet [NDJ 99].

2.2 Prise en compte d'un modèle de la partie opérative

Le modèle automate obtenu à partir du Grafcet présente le comportement libre de la commande spécifiée par Grafcet. Ce modèle calculé ne tient pas compte de l'écart entre la partie opérative et le comportement effectif de la commande, ce qui peut engendrer des effets directs sur les propriétés du modèle. En effet, sur le modèle du comportement libre, des blocages peuvent être identifiés et peuvent ne pas correspondre aux blocages réels, ce qui nécessite une attention particulière lors de la démarche de synthèse. Autrement dit, une prise en compte de la partie opérative permet de sensibiliser les transitions conduisant à une situation de blocage sur le comportement libre de façon à rendre cette situation non atteignable. D'autre part, même si la commande est identifiée sans blocage lors de ses évolutions théoriquement possibles, des blocages durant l'exécution réelle peuvent apparaître, si la commande entraîne la partie opérative dans une situation incompatible avec l'expression logique des transitions validées.

Lors de la conception de la commande, le concepteur tente souvent inconsciemment d'intégrer une image implicite de la partie opérative, représentant son comportement, dans le Grafcet afin d'exprimer au niveau de la commande, la causalité entre l'envoi des ordres et les réactions conséquentes de la partie opérative. Cependant, cela suppose que toutes les réactions de la PO et leur ordonnancement, soient connues de manière précise pour chaque action et chaque situation de la commande. En plus, la complexité et la nature parallèle des systèmes de commande et des procédés à commander entraînent des évolutions parallèles et complexes entre l'envoi des ordres et les réactions conséquentes du procédé. Toutes ces raisons font que l'image de la partie opérative n'est pas si simple à intégrer. Il faut donc s'appuyer sur un modèle explicite de la partie opérative, séparément de la commande.

La définition de ce modèle de partie opérative reste une tâche très délicate, en particulier lorsque le système a une taille importante et surtout si son comportement correspond aux évolutions d'un système physique de nature asynchrone et non déterministe, associant généralement des évolutions discrètes, des lois physiques linéaires ou non, probabilistes, temporisées, etc.... En effet, la modélisation de la partie opérative en vue de la synthèse de commande demande une analyse au préalable pour

résoudre des problèmes méthodologiques de structuration et de composition des éléments et choisir le degré de la granularité du comportement modélisé de ces éléments [ZAY 99]. Cette modélisation doit donc s'effectuer suivant un compromis entre l'objectif à atteindre, la précision du modèle à élaborer et la complexité des calculs résultants. Il est à noter que plus la granularité de la modélisation est fine, plus le calcul correspondant sera long et complexe [ZAY 01].

2.3 Synthèse automatique de la commande : Démarche d'implantation d'une commande correcte à partir d'une spécification Grafcet

Peu de travaux s'intéressent au passage de la spécification à l'implantation de la commande en tenant compte de la différence entre la sémantique du modèle de départ (spécification) et le modèle de la commande en exécution, d'où l'intérêt de développer dans cet objectif des méthodes respectant au mieux la réactivité et le déterminisme de la commande.

Pour passer d'une spécification de la commande par Grafcet à son implantation dans un API et pour que cette commande soit réactive, déterministe et sans blocage tout en respectant un certain nombre de contraintes de sûreté et de vivacité, [ZAY 01] a identifié quatre étapes nécessaires, présentées par le diagramme de la figure 2.2.

La *première étape* consiste à obtenir à partir du Grafcet de spécification, un modèle déterministe et réactif en vue de son implantation. L'automate ASS identifié précédemment dans l'étude comportementale du Grafcet, répond à ces critères.

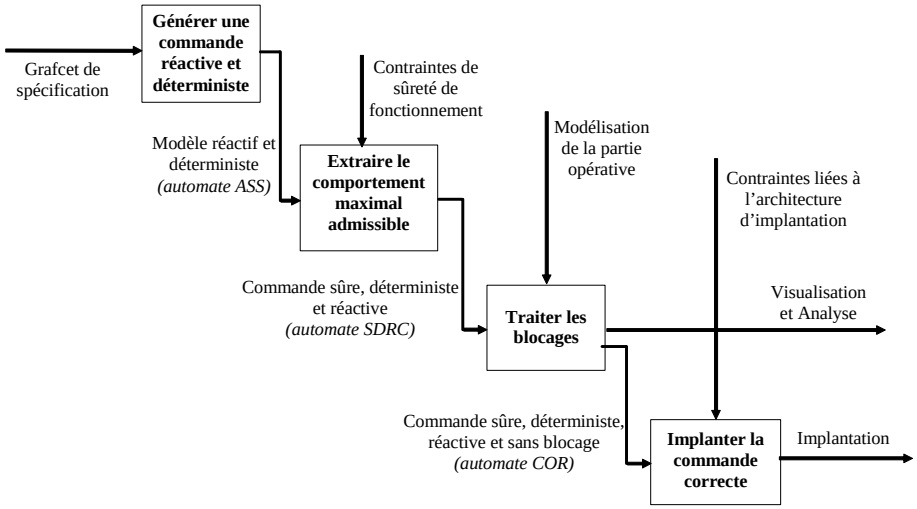


Figure 2.2. Les différentes étapes pour passer de la spécification à l'implantation.

La *deuxième étape* concerne l'intégration des contraintes dans la commande pour aboutir à un comportement le plus permissif de l'ASS vis-à-vis de ces contraintes. Ces contraintes consistent globalement à inhiber des ordres et/ou agencer des séquences d'exécution des commandes envoyées au procédé.

La *troisième étape* identifiée consiste à traiter les évolutions irréalistes et les blocages qui peuvent apparaître lors de l'exécution réelle à travers la prise en compte d'une modélisation adaptée de la partie opérative (Automate *COR*). Ce traitement s'effectue de façon automatique. Pour traiter ces blocages, il faut chercher à interdire certaines réactions de la partie opérative en agissant sur les actions appropriées du Grafcet. En effet, il est possible d'interdire les évolutions qui conduisent à un blocage, en inhibant les actions qui se situent dans les étapes en amont du blocage [ZAY 01]. Une telle approche s'apparente d'ailleurs à la synthèse du comportement commandable maximal selon Ramadge et Wonham [RAM 89].

La *quatrième et dernière étape* identifiée correspond à la phase d'implantation de la commande correcte et exempte de blocage (automate *COR*) dans une architecture programmable qui peut être mono-processeur ou distribuée [KOU 96].

Ces quatre étapes permettant le passage d'une spécification exprimée en Grafcet à son implantation ont été formalisées dans le cadre des travaux de thèse de Constant Ndjab [NDJ 99]. Ces travaux bénéficient des acquis de la théorie de supervision des SED pour élaborer une démarche permettant la construction systématique de la restriction minimale et sans blocage des évolutions d'un Grafcet donné, par rapport aux contraintes imposées de sécurité et vivacité.

3 SYNTHESE D'UNE IMPLANTATION OPTIMALE DANS LE CADRE DE LA THEORIE DE SUPERVISION [NDJ 99]

3.1 Présentation

Le travail effectué dans le cadre de la thèse de Constant Ndjab a ainsi abouti au développement d'une approche globale de synthèse et d'implantation d'une commande optimale à partir d'un Grafcet, d'un modèle du procédé à commander et des contraintes à respecter sur les évolutions du procédé [NDJ 99]. Cette approche a permis ainsi d'utiliser les facilités de modélisation du Grafcet, sans restrictions particulières, et de profiter des apports de l'ensemble des travaux liés à la théorie de supervision.

Pour bénéficier du cadre formel lié à la théorie de supervision [RAM 89], la modélisation de la partie opérative s'effectue sous forme d'automates décrivant les évolutions physiquement possibles 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 [BAL 93], 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'entrée du Grafcet avec le caractère événementiel du modèle de la théorie. Ainsi, nous avons retenu la correspondance suivante : 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. Les ensembles d'événements $\uparrow$Z et $\downarrow$Z, correspondent aux activations et désactivations des ordres, tandis que les ensembles d'événements $\uparrow$E et $\downarrow$E reflètent les fronts montants et descendants des entrées. Les ensembles Σ_c et Σ_u s'écrivent alors $\Sigma_c = \uparrow Z \cup \downarrow Z$ et $\Sigma_u = \uparrow E \cup \downarrow E$.

3.2 Démarche de synthèse

La démarche de synthèse, comprend les six étapes suivantes (figure 2.3) :

1- Modélisation.

2- Synthèse du superviseur.

3- Extraction de l'automate *ASS*.

4- Intersection entre l'ASS et le superviseur.

5- Réduction.

6- Exécution de la commande.

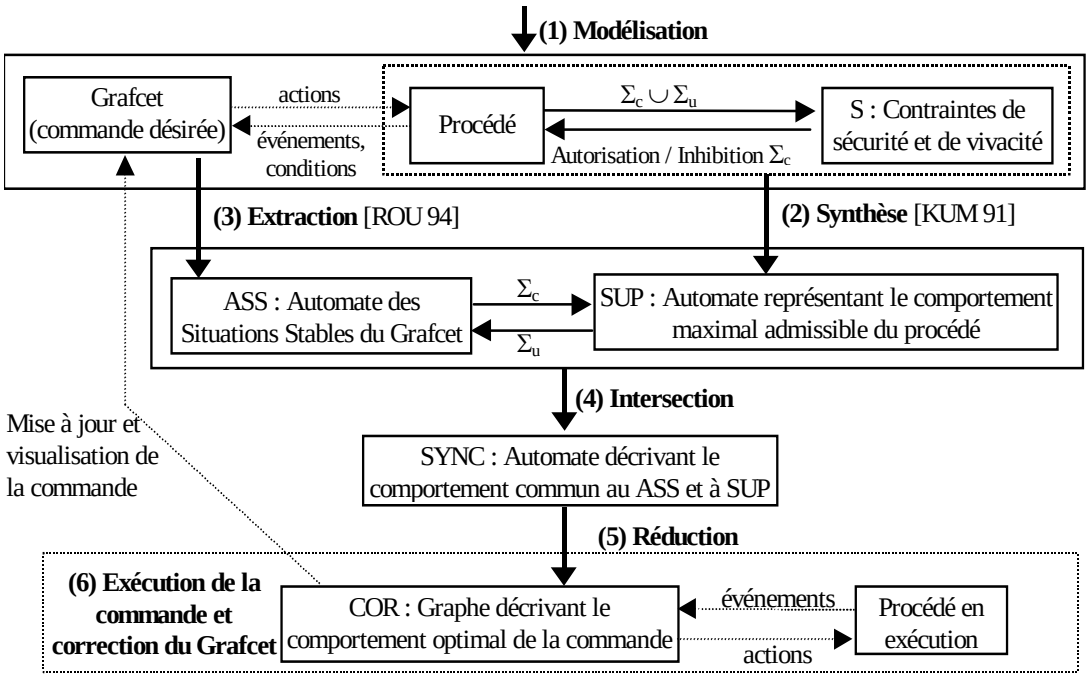


Figure 2.3. Les étapes de l'approche de synthèse hors ligne.

Les différents algorithmes formels, sont donnés dans l'article [CAR 01]

1- L'étape de modélisation est décomposée en trois phases :

(i) l'élaboration du Grafcet correspondant à la spécification de la commande,

(ii) la modélisation de la partie opérative sous forme d'automates conformes à ceux utilisés dans la théorie de supervision. Ces automates décrivent les évolutions physiquement possibles du procédé par des événements simples.

(iii) la spécification des contraintes de sécurité et de vivacité sous forme d'automates conformes au modèle de la théorie de supervision.

2- L'opération de synthèse consiste à partir des contraintes et du procédé modélisés précédemment, à générer l'automate du superviseur *SUP* selon l'algorithme de synthèse proposé par [KUM 91]. Cet automate correspond au comportement commandable maximal admissible du procédé par rapport aux contraintes, et il est décrit par un 4-uplet $S = (\Sigma, Q, \Delta, q_0)$, avec Σ l'ensemble des événements, Q l'ensemble des états, q_0 l'état initial et Δ une fonction de transition partielle $\Delta: Q \times \Sigma \rightarrow Q$. Une transition est définie par un triplet (q, σ, q') tel que $\Delta(q,\sigma)=q'$, q étant l'état de départ, σ l'événement correspondant à la transition et q' l'état d'arrivée.

3- L'extraction consiste à appliquer l'algorithme de [ROU 94] afin de traduire le Grafcet en automate équivalent noté *ASS* (Automate des Situations Stables). L'automate *ASS* est décrit par un 5-uplet (E, Z, X, T, x_0), avec :

- E l'ensemble des entrées du Grafcet,

- Z l'ensemble des sorties du Grafcet,

- X l'ensemble des états représentant les situations stables du Grafcet. A chaque état x de X on associe les deux ensembles suivants :

  - $Z_x \subseteq Z$ qui représente toutes les sorties actives dans cette situation.

  - $Etapes_x$ qui représente toutes les étapes du Grafcet actives dans cette situation.

- T: X x f(E) $\rightarrow$ X est une fonction partielle représentant les transitions de ASS qui correspondent aux évolutions du Grafcet à l'échelle du temps externe.

- x_0 est l'état initial.

4- C'est l'étape clé de cette démarche qui consiste à générer un automate appelé *SYNC*. Cet automate décrit le comportement commun entre les automates *SUP* et *ASS* de manière à ne retenir, dans le modèle de commande, que les évolutions autorisées par le superviseur. Le principe d'intersection élaboré est assez particulier car chacun des automates *ASS* et *SUP* repose sur une sémantique propre. Pour l'automate *ASS*, chaque état est muni d'un ensemble d'actions actives lors de la situation correspondante du Grafcet, et les transitions sont décrites par une expression logique composée des entrées

du Grafcet et de fronts sur ces entrées. Quant à *SUP*, il correspond à un automate rudimentaire où les transitions sont décrites par des événements.

L'automate *SYNC* se construit en tant que modèle asynchrone dont les transitions correspondent à des événements simples. Les ordres qui doivent être émis en parallèle dans une situation du Grafcet, sont représentés dans *SYNC* par une structure arborescente composée de transitions décrivant les activations et les désactivations successives autorisées par le superviseur. Les états reliés par ces transitions constituent une région correspondante à la dite situation du Grafcet. Ainsi, l'automate *SYNC* est constitué d'un ensemble de régions (figure 2.4). Les transitions intra-région correspondent aux événements commandables, et les transitions inter-régions aux événements non commandables.

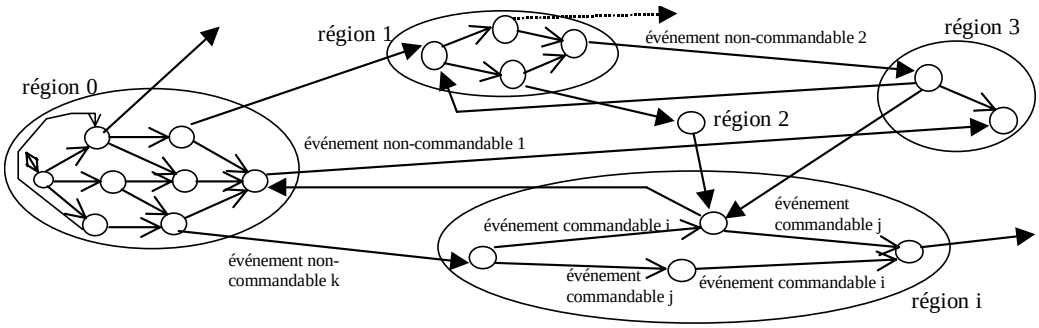


Figure 2.4. Modèle sémantique de l'automate *SYNC*

Formellement, l'automate SYNC est défini par un 6-uplet (Σ, ST, TR, st_0, r_0, BLOC) avec :

- $\Sigma = \Sigma_c \cup \Sigma_u$, où $\Sigma_c = \uparrow Z \cup \downarrow Z$ et $\Sigma_u = \uparrow E \cup \downarrow E$. Dans la suite, $\uparrow z$ correspond à un élément de $\uparrow Z$, $\downarrow z$ à un élément de $\downarrow Z$, $\uparrow e$ à un élément de $\uparrow E$ et $\downarrow e$ à un élément de $\downarrow E$.

- ST est l'ensemble des états de SYNC partitionné en sous-ensembles appelés régions. Un état $st \in ST$ correspond à une configuration unique du 3-uplet (q, y, E), où *q* est un état de S, *y* un état de ASS et *E* l'ensemble des valeurs logiques (0 ou 1) des entrées du Grafcet. Tous les états correspondant à une situation du Grafcet et à une configuration unique de ses variables d'entrée, sont groupés dans une région. Chaque région $r \subset ST$ est munie de l'ensemble $Etapes_r$ qui contient les étapes actives dans la situation correspondante du Grafcet.

- TR: $ST \times \Sigma \rightarrow ST$ est une fonction partielle représentant les transitions de SYNC. Une transition est définie par un triplet $(st, \sigma, st') \in TR$.

- $st_0 \in ST$ est l'état initial qui est donné par (q_0, y_0, E_0), où q_0 est l'état initial de S, y_0 l'état initial de ASS et E_0 correspond à l'ensemble des valeurs initiales des entrées du Grafcet.

- $r_0 \subset ST$ est la région initiale correspondant à la situation initiale y_0 du Grafcet et aux valeurs initiales E_0 de ses variables d'entrée. Cette région, qui inclut l'état initial st_0 est munie de l'ensemble des étapes correspondantes à la situation initiale du Grafcet $Etapes_{r0} = Etapes_{y0}$.

- BLOC $\subset ST$ contient l'ensemble des états bloquants, c'est à dire ceux n'ayant pas de transitions sortantes. Dans chacun de ces états, les actions et/ou les évolutions de la commande sont en contradiction avec les événements issus des états correspondants du superviseur.

5- Cette étape consiste à réduire l'automate *SYNC* pour obtenir l'automate *COR* et s'effectue en trois phases :

(i) Les états bloquants de *SYNC* sont rendus non atteignables par suppression de leurs transitions amont.

(ii) La phase dite d'optimisation consiste à retirer de l'automate *SYNC* résultant de la première phase, les transitions qui sont non atteignables durant l'exécution réelle.

(iii) La dernière phase procède à l'agrégation des situations instables, de manière à aboutir à une commande réactive qui évolue, suite à l'occurrence d'un événement non commandable, d'une situation stable à une autre.

6- L'automate résultant de l'étape de réduction (*COR*) est ensuite implanté sur **PC** [ZAY 99] ou sur Automate Programmable Industriel (**API**) [CAR 00] [ZAY 97] sous forme d'une table. Il correspond à la restriction minimale du comportement du Grafcet qui garantit la conformité par rapport aux contraintes spécifiées et le non blocage du système. L'exécution de la commande optimale implantée (*COR*) permet la visualisation des corrections apportées au Grafcet original.

3.3 Discussion

Le travail effectué dans le cadre de la thèse de C. Ndjab [NDJ 99] 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.

Le contexte formel de la théorie de supervision des SED a été retenu en vue de démontrer la faisabilité de l'approche en faisant dans un premier temps, abstraction des difficultés d'utilisation.

Cette faisabilité ayant été prouvée sur plusieurs exemples simples [CAR 01], l'objectif de notre travail de thèse est de développer l'applicabilité d'une telle démarche à des cas industriels. Il s'agit donc de confronter la démarche à des exemples réels, de mettre à jour les difficultés d'utilisation et de proposer des solutions pour répondre à l'objectif fixé.

Par conséquent, la suite de ce chapitre est consacrée aux premiers travaux de thèse permettant de valider les travaux précédents sur un exemple réel, de formaliser définitivement la synthèse en ligne et d'indiquer les difficultés rencontrées en terme d'utilisation et d'applicabilité de la démarche. L'exemple présenté à la suite va servir d'illustration aux différents chapitres de la thèse.

4 CONTEXTE DES PREMIERS TRAVAUX DE THESE

Fin 2001, dans le cadre des activités du groupe COSED [1] devenu INCOS [2] et afin de favoriser la synergie entre travaux de recherche, il a été décidé de mettre en place un projet de recherche coopératif. L'objectif général de ce projet etait de favoriser le couplage entre les travaux des différentes équipes au travers d'une étude de cas, afin de :

- Pouvoir comparer les résultats de recherches
- Etablir des connexions entre les différentes approches
- Susciter de nouveaux thèmes de recherche.

La mise en place de ce projet a conduit à définir le thème « Modélisation de la partie opérative pour la synthèse et l'analyse de la commande ». Au sein de ce thème, il a été décidé de se doter d'un cas d'étude commun sur lequel chaque équipe pourrait appliquer ses techniques. Ce support applicatif, benchmark potentiel, est un ensemble opératif issu d'une machine d'assemblage réelle (figure 2.5). Cette machine d'assemblage est intallée au laboratoire d'automatique du Département de Génie Mécanique de l'ENS Cachan.

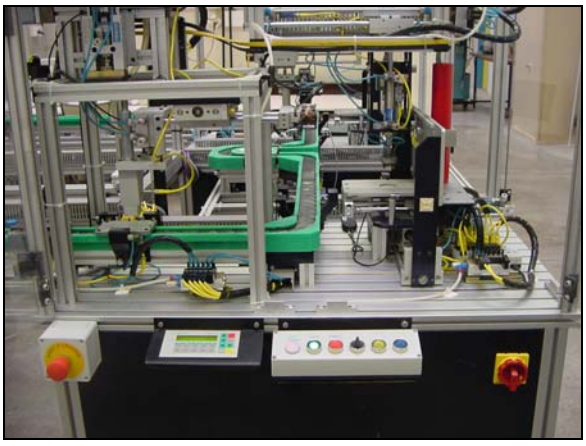


Figure 2.5. Benchmark du groupe de travail INCOS

[1] COSED = Commande opérationnelle des SED.

[2] INCOS = Ingénierie de la commande et de la supervision des SED.

- Modélisation de la partie opérative pour la synthèse et l'analyse de la commande

L'utilisation d'un modèle de partie opérative pour la synthèse et l'analyse de la commande est une nécessité exprimée tant par les industriels que par la communauté des chercheurs SED. Pour les industriels, il s'agit avant tout d'utiliser ce modèle de partie opérative pour réaliser une simulation permettant de vérifier hors ligne les programmes de commande, et aussi pour concevoir des interfaces pour les opérateurs de conduite. On trouve d'ailleurs sur le marché plusieurs progiciels développés à ces fins.

La recherche en SED s'est intéressée quant à elle à la modélisation de la partie opérative dans deux domaines :

- **Synthèse de la commande.** Il s'agit alors de générer un modèle de commande respectant des propriétés de sûreté exprimées relativement à la partie opérative, et nécessitant donc un modèle de cette dernière.
- **Analyse de la commande, et plus particulièrement preuves de propriétés sur un modèle de commande.** L'expression formelle des propriétés de vivacité, sécurité nécessite un modèle, implicite ou explicite, de la partie opérative.

- Approches proposées

Les travaux du groupe ont donné lieu à des communications proposées au congrès sur la Modélisation des Systèmes Réactifs (MSR 03). Les articles proposés ont cherché à montrer l'intérêt de la prise en compte d'un modèle de partie opérative pour assurer le bon fonctionnement du procédé.

Pour Gouyon et al [GOU 03], la résolution des problèmes de modélisation identifiés au paragraphe précédent repose sur la mise en œuvre d'une approche modulaire de synthèse et plus précisément sur la synthèse d'une commande hiérarchisée dont la structure se base sur une décomposition modulaire des processus physiques à commander pour aboutir à des modèles « ultimes » de procédé et de spécifications à base d'automates.

Pour Roussel et al. [ROU 03], la modélisation de la partie opérative se base sur une formalisation algébrique des relations liant les variables-clés du programme (entrées/sorties du système de contrôle commande, variables reflétant l'état de la partie

opérative commandé). L'intérêt de cette méthode de synthèse formelle d'un programme sûr de contrôle commande réside dans l'utilisation d'une alternative intéressante aux travaux basés sur la théorie de la commande supervisée, [RAM 87] en exprimant les spécifications par assertions algébriques en lieu et place d'automates à états, ce qui prévient de tout problème d'explosion combinatoire.

Dans l'article publié par D. Gaffé [GAF 03], la conception des systèmes de commande est basée sur une réalisation modulaire et hiérarchique et plus particulièrement sur une approche modulaire synchrone par le biais du modèle SYNCCHARTS issu de la famille des STATECHARTS [HAR 87]. L'idée consiste à modéliser d'abord la *Partie Commande* du système, puis intégrer progressivement un modèle de la *Partie Opérative* en s'appuyant sur l'expressivité de SYNCCHARTS pour définir de multiples comportements locaux de cette PO.

Le quatrième travail présenté dans ce congrès, [PHI 03a] [TAJ 03], concerne la méthode de synthèse qui a été traitée dans le reste du chapitre

Malgré la qualité très grande de ces travaux, on peut remarquer que les résultats s'avèrent encore insuffisants pour permettre leur utilisation dans tous les cas de figure industriels. Les principales critiques concernent la trop grande simplicité des modèles de partie opérative utilisés et la non structuration de la modélisation.

5 MISE EN ŒUVRE DE LA DEMARCHE

Pour faire le point sur la démarche de synthèse hors ligne et comparer avec ceux de la communauté, nous avons choisi le benchmark introduit dans le cadre du groupe de travail COSED.

5.1 Description du benchmark

L'exemple est constitué d'un préhenseur pneumatique présenté figure 2.6a, son rôle est de saisir une pièce (pignon) dans un tiroir pour ensuite le déposer sur un axe. La préhension s'effectue par aspiration, les déplacements se font par vérins double effet pilotés par un distributeur monostable pour le mouvement vertical, afin d'avoir une remontée lorsque l'ordre *DESCENDRE* n'est plus commandé, et par un distributeur bistable pour le mouvement horizontal. La figure 2.6b présente l'ensemble des entrées/sorties du système.

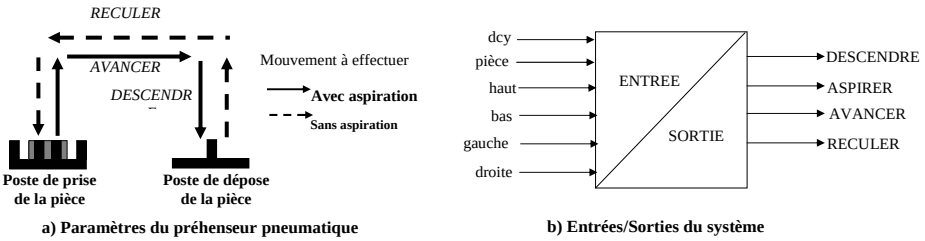


Figure 2.6. Préhenseur pneumatique.

Le vecteur d'entrée est défini par E= {gauche, droite, haut, bas, dcy, pièce} et le vecteur de sortie par Z= {AVANCER, RECULER, DESCENDRE, ASPIRER}.

a. Etape de modélisation

- *Extraction de l'automate des situations stables ASS*

La figure 2.7 présente le Grafcet de commande (figure 2.7a) en fonctionnement normal du préhenseur et son automate équivalent ASS (figure 2.7b).

Description du Grafcet : le déclenchement du cycle est effectué par un bouton poussoir *dcy*. Le mouvement vertical peut alors commencer par l'envoi de l'ordre *DESCENDRE* jusqu'à obtenir *bas*. L'aspiration de la pièce est donc possible en maintenant constamment la descente (étape 2 du Grafcet). La détection du capteur pièce

permet d'activer l'étape 3 où les deux actions *ASPIRER* et *AVANCER* sont envoyées. L'activation de l'ordre *AVANCER* et la suppression de *DESCENDRE* permet alors d'effectuer un mouvement diagonal, donc deux positions peuvent être atteintes : soit arriver à la position haute avant droite (étape 3 puis étape 4) ou arriver à droite avant haut (étape 3 avant étape 5). A partir de l'étape 4 ou l'étape 5 et suite au franchissement de la transition *droite* ou de la transition *haut*, l'étape 6 devient active et le mouvement vertical en aspirant est déclenché pour atteindre la position de dépose en bas (*bas*). De cette position, en maintenant toujours la descente (*DESCENDRE*), l'aspiration est arrêtée (étape 7) jusqu'à obtenir ↓pièce. L'activation de l'étape 8 provoque le retour du préhenseur vers la position initiale en effectuant un mouvement diagonal en passant soit par l'étape 10 si on arrive à gauche avant haut ou soit par l'étape 9 si le préhenseur arrive en haut avant gauche.

L'outil AGGLAE [ROU 94] délivre un automate *ASS* composé de 11 états et 23 transitions, cet ASS est présenté par la figure 2.7b.

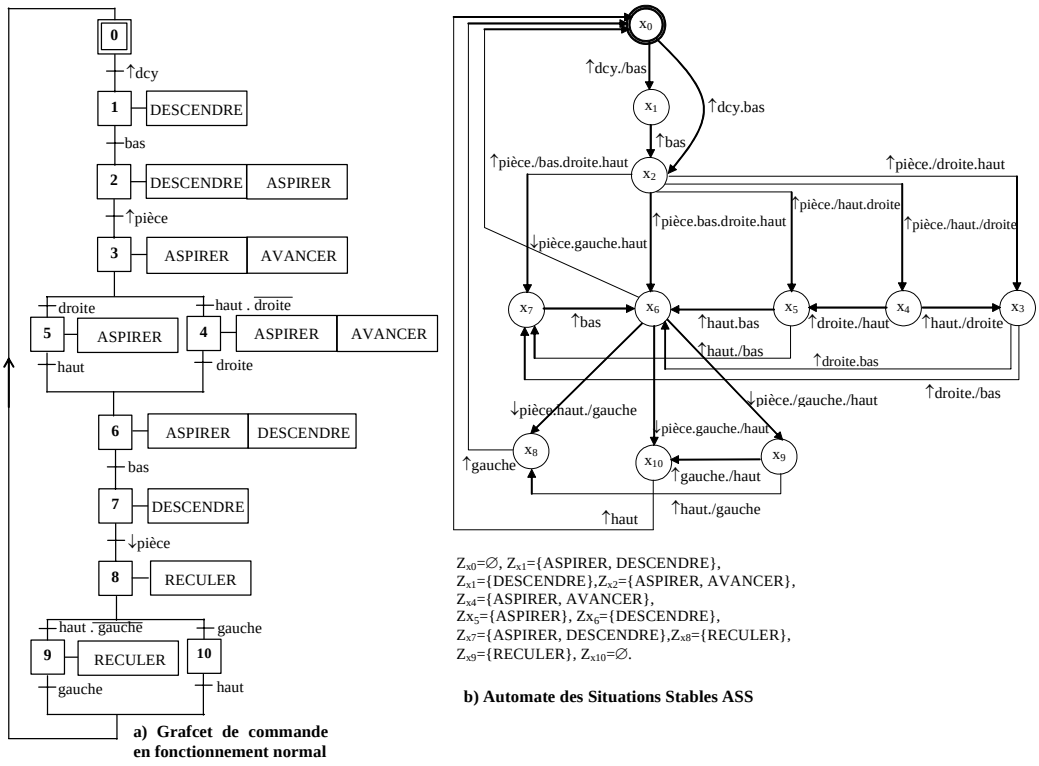


Figure 2.7. Extraction de l'automate ASS.

- *Modélisation intuitive de la partie opérative*

Le modèle élaboré est constitué de quatre automates décrivant respectivement les chaînes fonctionnelles du mouvement horizontal (Figure. 2.8a), du mouvement vertical (Figure. 2.8b), du mouvement d'aspiration (Figure. 2.8c) et du bouton poussoir de départ cycle (Figure. 2.8d). Ces modèles tiennent compte de la technologie des vérins qui animent le préhenseur pneumatique. En situation initiale, le préhenseur est en position haute, à gauche ou à droite, l'aspiration est au repos et le bouton poussoir *dcy* est relâché.

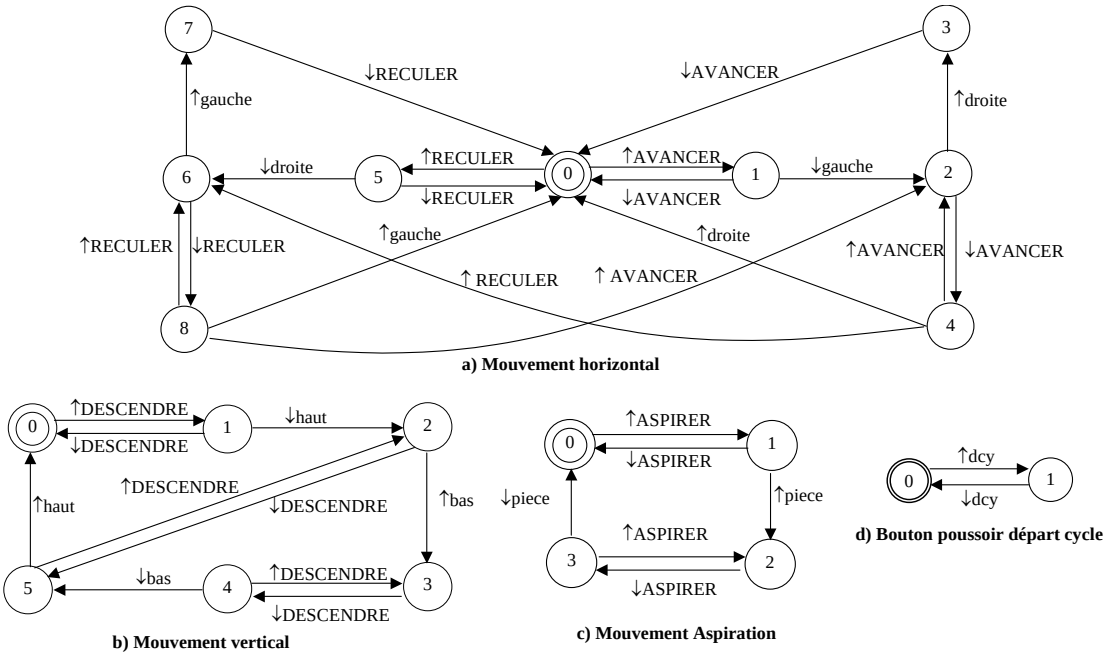


Figure 2.8. Modèle de la partie opérative.

Partant de sa position initiale (état 0), le préhenseur (Figure. 2.8a) peut se déplacer à droite (état 1) ou à gauche (état 5). L'activation de l'ordre *AVANCER* lance le déplacement du préhenseur vers la droite, lui faisant quitter la position *gauche* (état 2). La continuation du mouvement lui fait atteindre la position *droite* (état 3). La désactivation de l'ordre *AVANCER* lui permet de revenir à l'état initial. A partir de l'état 1, si l'ordre *AVANCER* est désactivé, le préhenseur retourne à son état initial. Dans l'état 2, la désactivation de *AVANCER* ne permet plus un retour en situation initiale car la technologie utilisée force le préhenseur à aller vers la droite (état 4 puis état 0). Le comportement du préhenseur pour un mouvement vers la gauche à partir de la situation initiale est similaire à celui de droite.

En raison de la technologie associée au mouvement vertical (Figure 2.8b) défini par

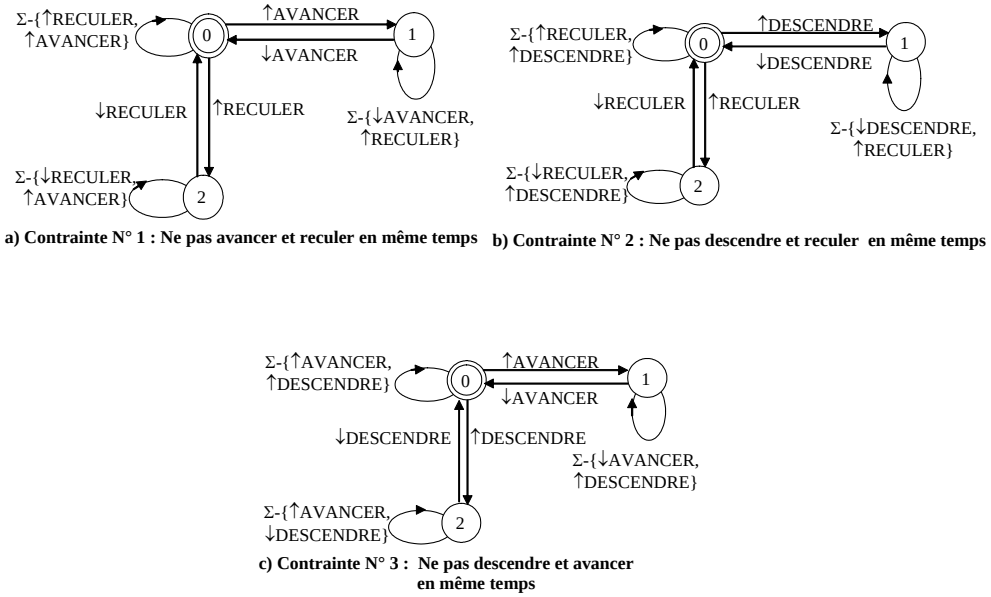
un distributeur de vérin simple effet, le préhenseur ne peut recevoir à partir de son état initial (état 0), que l'ordre *DESCENDRE* (état 1). En recevant cet ordre, le préhenseur quitte la position haute (état 2) pour atteindre la position basse (état 3) si le mouvement de descente n'a pas été interrompu par la désactivation de l'ordre (état 5). La désactivation de *DESCENDRE* (état 4) entraîne la remontée du préhenseur pour atteindre le fin de course *haut* et revenir à l'état initial.

L'aspiration (Figure. 2.8c) est modélisée par un automate à quatre états. L'état initial est quitté à réception de l'ordre *ASPIRER* (état 1). L'événement *pièce* confirme l'aspiration de la pièce (état 2). Lorsque l'ordre *ASPIRER* est inhibé (état 3), la désactivation de *pièce* entraîne le retour à l'état initial. A partir des états 1 et 3, l'aspiration peut être inhibée ou relancée. Enfin, le bouton poussoir de départ cycle est soit au repos, soit appuyé, ce qui se traduit par un automate à deux états (Figure. 2.8d).

Le modèle global de la partie opérative est obtenu par la composition asynchrone de ces différents automates et possède pour cet exemple 432 états et 2664 transitions. Nous pouvons constater que le nombre d'états et de transitions est assez important au regard de la simplicité de cet exemple.

- *Spécification des contraintes*

Pour cet exemple, nous imposons quatre contraintes de sécurité (Figure. 2.9.a, b, c et d) ainsi que deux contraintes de vivacité liées à l'aspiration (Figure. 2.9e et 2.9f). La première contrainte (Figure. 2.9a) consiste à interdire l'activation simultanée des ordres *AVANCER* et *RECULER*. L'état 0 correspond à l'ensemble des états du procédé où tous les ordres sont autorisés sauf *AVANCER* et *RECULER*. L'évolution vers l'état 1 s'effectue par l'activation de l'ordre *AVANCER* (le retour à l'état 0 est conditionné par la désactivation de *AVANCER*). L'état 1 autorise alors tous les ordres, excepté l'ordre *RECULER*. Pour interdire l'activation de l'ordre *AVANCER*, quand l'ordre *RECULER* est envoyé vers la partie opérative (état 2), le raisonnement est identique. Cette modélisation s'applique aux contraintes n°2 (Figure. 2.9b) et n°3 (Figure 2.9c) interdisant l'occurrence simultanée des ordres *DESCENDRE* et *RECULER*, et *AVANCER* et *DESCENDRE*.



a) Contrainte N° 1 : Ne pas avancer et reculer en même temps

b) Contrainte N° 2 : Ne pas descendre et reculer en même temps

c) Contrainte N° 3 : Ne pas descendre et avancer en même temps

Figure 2.9 : Contraintes de sécurité.

Pour des raisons liées à la sécurité du personnel et à la saisie du pignon sur son poste, la quatrième contrainte étudiée (Figure. 2.10) n'autorise les mouvements horizontaux qu'en position haute. A partir de l'état initial, tous les événements sont autorisés hormis les activations de *RECULER*, *AVANCER* et *haut*. L'interception du front montant *haut* fait changer l'automate d'état (état 1). Dans cet état 1, les ordres *AVANCER* et *RECULER* peuvent être envoyés vers la partie opérative.

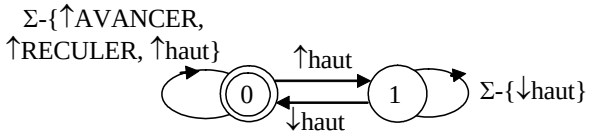
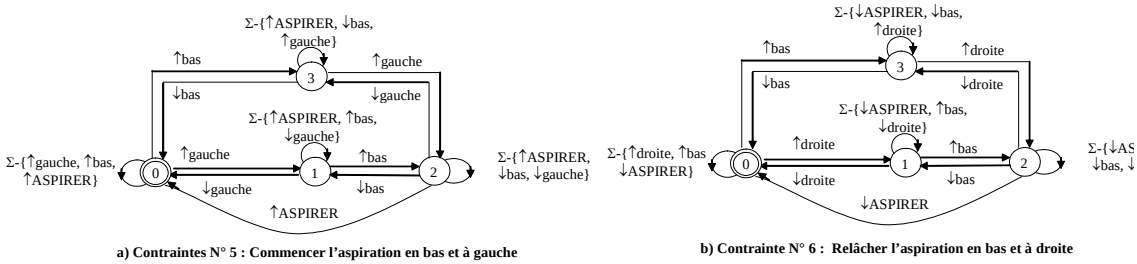


Figure 2.10. Contrainte N° 4 : Avancer ou reculer en position haute.

Les automates des figures 2.11.a et 2.11.b décrivent deux contraintes de vivacité qui représentent la chronologie des événements autour de l'aspiration. Pour la contrainte n°5, l'ordre *ASPIRER* est envoyé lorsque le préhenseur se situe en bas à gauche et inhibé quand il est en bas à droite pour la contrainte n°6.



a) Contraintes N° 5 : Commencer l'aspiration en bas et à gauche

b) Contrainte N° 6 : Relâcher l'aspiration en bas et à droite

Figure 2.11. Modèle des contraintes de vivacité.

Le modèle global des contraintes globales est obtenu par composition synchrone des contraintes. Il se compose de 128 états et 1856 transitions. L'opération de synthèse génère un automate *SUP* qui correspond au comportement commandable maximal admissible du procédé par rapport aux contraintes, comprend 1030 états et 4143 transitions. Des extraits de *SUP* sont proposés figure 2.12.b.

b. Etape de génération de la commande : Synthèse hors ligne

➢ *Génération du comportement commun*

L'objectif maintenant est de déterminer le comportement commun des automates *SUP* et *ASS,* pour obtenir l'automate *SYNC,* de manière à ne retenir, dans le modèle de commande, que les évolutions autorisées par le superviseur *SUP*.

La construction complète de *SYNC* fournit un automate composé de 95 états et 187 transitions, regroupés à travers 57 régions (figure 2.12). Les premiers éléments de cet automate sont présentés dans la figure 2.12.c avec pour ensemble des entrées du Grafcet,

E = {gauche, droite, haut, bas, dcy, pièce}. La région r_0 représente la région initiale de cet automate. Cette région est développée à partir de l'état initial défini par le triplet *(q_0, y_0, E_0)* où q_0 représente l'état initial de *S* (figure 2.12 b), y_0 l'état initial de *ASS* (figure 2.12 a) et E_0 = {gauche, /droite, haut, /bas, /dcy, /pièce}. Les sorties actives à l'instant initial sont données par l'ensemble Z_{y0} avec $Z_{y0}=\varnothing$. Aucun événement commandable ne se produit dans la situation y_0 de *ASS*, la région initiale ne comporte qu'un seul état. Il faut alors créer à partir de *(q_0, y_0, E_0)*, les transitions de sortie inter-régions supportées par des événements non commandables.

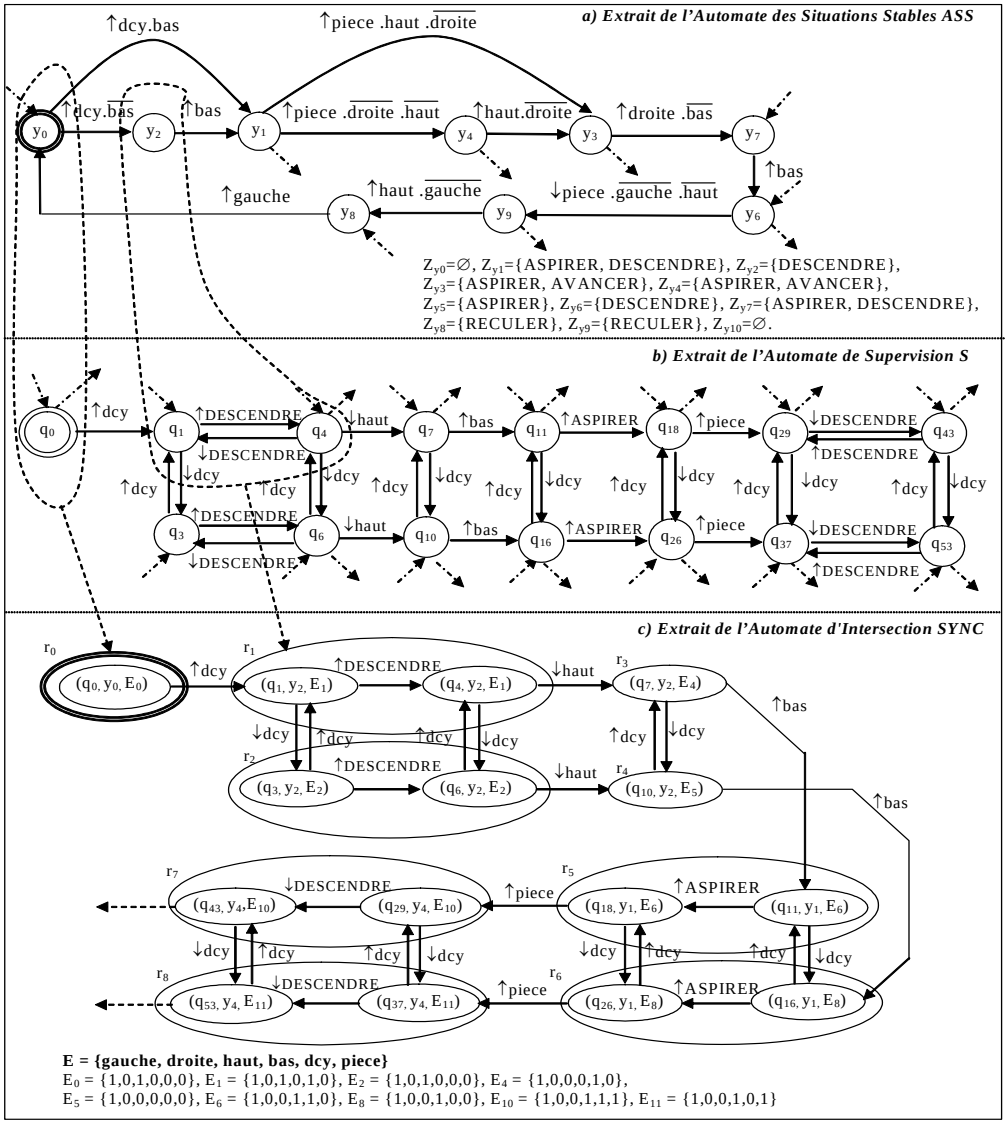


Figure 2.12. Extrait des automates ASS, SUP et SYNC.

L'une des trois transitions sortantes de q_0 est l'événement non commandable $\uparrow dcy$ et comme *dcy* est égal à 0 dans l'état y_0, le front montant $\uparrow dcy$ peut se produire dans l'état (q_0, y_0, E_0). Cet événement valide l'expression logique de la transition reliant y_0 à y_2 dans *ASS* et par conséquent, l'état initial de la nouvelle région r_1 est (q_1, y_2, E_1) avec E_1 = {gauche, /droite, haut, /bas, dcy, /pièce}. Comme $\uparrow DESCENDRE$ représente l'unique événement commandable se produisant dans la situation courante y_2 de *ASS* et qu'il est autorisé dans la situation courante q_1 du superviseur *S*, r_1 inclut une transition correspondant à l'occurrence de $\uparrow DESCENDRE$, reliant le nouvel état (q_4, y_2, E_1). A partir de chaque état de cette région r_1, seront déterminées les transitions sortantes supportées par des événements non commandables, ce qui permet de créer deux

nouvelles régions r_2 et r_3. On remarquera sur la figure 2.12c que plusieurs régions n'incluent qu'un seul état car elles correspondent à une situation où aucun nouvel ordre ne doit être activé ou désactivé (régions r_3 et r_4).

- *Génération de l'automate de commande*

L'objectif est maintenant de générer la commande réactive et déterministe (automate *COR*) correspondant au plus grand sous-ensemble de comportements de *SYNC*, exempt de blocage. L'automate de commande *COR* dont un extrait est présenté figure 2.13, comporte 57 états et 111 transitions.

Sur la figure 2.13, à chaque état de *COR* est associé le quadruplet (Z_c, Sit_c, ORD, PROH) où Z_c correspond à l'ensemble des sorties du Grafcet lié à cet état et Sit_c l'ensemble des étapes Grafcet correspondant à cet état. *ORD* et *PROH* représentent l'ensemble des ordres, calculés dynamiquement, à envoyer, respectivement à interdire, par *COR*. Ces ensembles sont utilisés pour renseigner l'utilisateur sur les ordres qui sont maintenus ou non par la procédure de synthèse. La région r_0 correspond désormais à l'état 0 de *COR* où aucun ordre n'est envoyé. Les deux états de la région r_1 sont agrégés en un unique état, l'état 1 où l'ordre envoyé pendant l'exécution est *DESCENDRE*, aucun ordre n'étant interdit.

Deux corrections induites par la procédure de synthèse sont présentées dans la figure 2.13. Ces corrections se manifestent par une restriction imposée et relative à la contrainte n°4 qui n'autorise les translations à droite ou à gauche qu'en position haute. Dans la situation {2} du Grafcet correspondant à l'état 6 de *COR*, les ordres envoyés sont *DESCENDRE* et *ASPIRER*. Dès que la pièce est saisie ($\uparrow$*pièce*), la situation Grafcet {3} devient active (état 8 dans *COR*). Dans cet état, les ordres normalement envoyés par le Grafcet sont {*ASPIRER, AVANCER*} (cf Z_8) alors que l'unique ordre envoyé pendant l'exécution est donné par ORD= {*ASPIRER*} avec PROH = {*AVANCER*} ce qui signifie que l'ordre *AVANCER* est interdit dans l'état 8 de *COR*. Cette interdiction reste valable dans l'état 14 de *COR* et ce, tant que le préhenseur n'a pas atteint la position haute ($\uparrow$*haut*). Dès que cette position est atteinte (état 17 dans *COR*), l'ordre *AVANCER* est autorisé car ORD = {*ASPIRER, AVANCER*} et PROH=$\varnothing$.

Le raisonnement est identique dans la phase de retour du préhenseur. A partir de la situation {7} du Grafcet (état 33 dans *COR*) pour laquelle ORD = {*DESCENDRE*} et

PROH=∅, dès que la pièce est relâchée (↓*pièce*), la situation Grafcet {8} devient active (état 38 dans *COR*). Dans cette situation, ORD=∅ et PROH= {*RECULER*} ce qui implique que l'ordre *RECULER* est interdit dans l'état 38 de *COR* pendant la remontée du préhenseur. Cette interdiction reste valable dans l'état 42 de *COR* et ce, tant que le préhenseur n'a pas atteint la position haute (↑*haut*). Dès que cette position est atteinte (état 45 dans *COR*), l'ordre *RECULER* est envoyé car ORD= {*ASPIRER, AVANCER*} et PROH=∅.

Pour cet exemple, aucun blocage n'a été détecté et les corrections paraissent évidentes sur cet exemple. Il est certain que la contrainte n°4 où la translation n'est autorisée qu'en position haute aurait pu être intégrée directement au niveau du Grafcet de commande. Cependant, pour des systèmes plus importants, la complexité et la nature parallèle de la commande et des procédés à commander entraînent des évolutions parallèles complexes entre l'envoi des ordres et les réactions conséquentes du procédé, il devient donc difficile d'intégrer dans la commande et de manière systématique l'ensemble des contraintes liées à la sécurité et la vivacité.

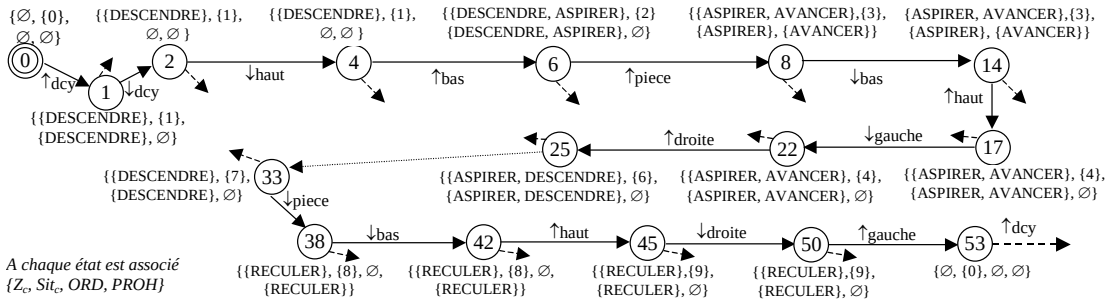


Figure 2.13. Extrait de l'automate de commande *COR.*

c. Discussion

Nous avons montré l'applicabilité d'une démarche de synthèse basée sur la théorie de supervision pour la commande en fonctionnement normal d'un préhenseur pneumatique. Le tableau 2.1 résume les tailles des différents automates. On constate que l'automate de commande *COR* est un automate beaucoup plus riche en informations que le simple automate ASS. En effet, il intègre par exemple des informations sur l'état de la partie opérative (↓*haut*, par exemple) et mémorise des ordres interdits ce qui n'est pas le cas pour l'automate ASS. Dans un contexte de supervision industrielle, par exemple,

l'implantation de cet automate permettra de renforcer les connaissances sur le système et ainsi de faciliter à l'opérateur humain, sa perception de l'état de l'installation.

Nature des Automates	ASS	Procédé	Contraintes	Superviseur S	SYNC	COR
Taille	11 états 23 trans	432 états 2664 trans.	128 états 1856 trans.	1030 états 4143 trans.	95 états 187 trans. 57 régions	57 états 111 trans.

Tableau 2.1 : Bilan des différents automates.

On peut rapidement noter que pour des systèmes complexes, l'utilisation de cette approche est confrontée au problème de la taille mémoire nécessaire pour l'implantation de l'automate de commande. On assiste à une explosion combinatoire du nombre d'états et à des problèmes d'espace mémoire. Pour pallier ce problème, les premiers travaux sur une approche formelle de synthèse similaire mais effectuée en ligne ont été établis [NDJ 99] et formalisés dans [TAJ 03]. Dans le paragraphe suivant, nous détaillons cette approche, sa formalisation est proposée en annexe 2.

5.2 Synthèse en ligne

5.2.1 Présentation

La démarche de synthèse de commande présentée précédemment est une démarche hors ligne. Nous proposons pour résoudre le problème de la taille de l'automate de commande, une modification de la démarche basée sur une technique de contrôle de fenêtre limitée comme proposée par Chung et al [CHU 92]. Ceci nous conduit à proposer une méthode de synthèse en ligne adaptée à notre démarche.

L'objectif de la synthèse en ligne ([PHI 03b], [TAJ 03], [PHI 04b]) est de réduire de manière significative le nombre d'états de l'automate de commande optimale en calculant la commande au fur et à mesure du fonctionnement du système. Par conséquent, la procédure de génération de la commande s'articule autour d'une étape d'intersection partielle des automates *SUP* et *ASS* pour quelques évolutions et d'une étape de traitement des blocages issus de cette intersection partielle afin de générer le

modèle de la commande à appliquer sous forme d'automate partiel qui anticipe le fonctionnement sur quelques pas de calcul.

La synthèse en ligne est basée sur trois étapes dont la première est la modélisation. Suite à l'extraction de l'ASS et suite à la synthèse du comportement commun SUP entre les contraintes et la PO, une intersection partielle consistant à calculer le comportement commun entre l'ASS et le SUP pour quelques pas d'évolutions (figure 2.14), est effectuée.

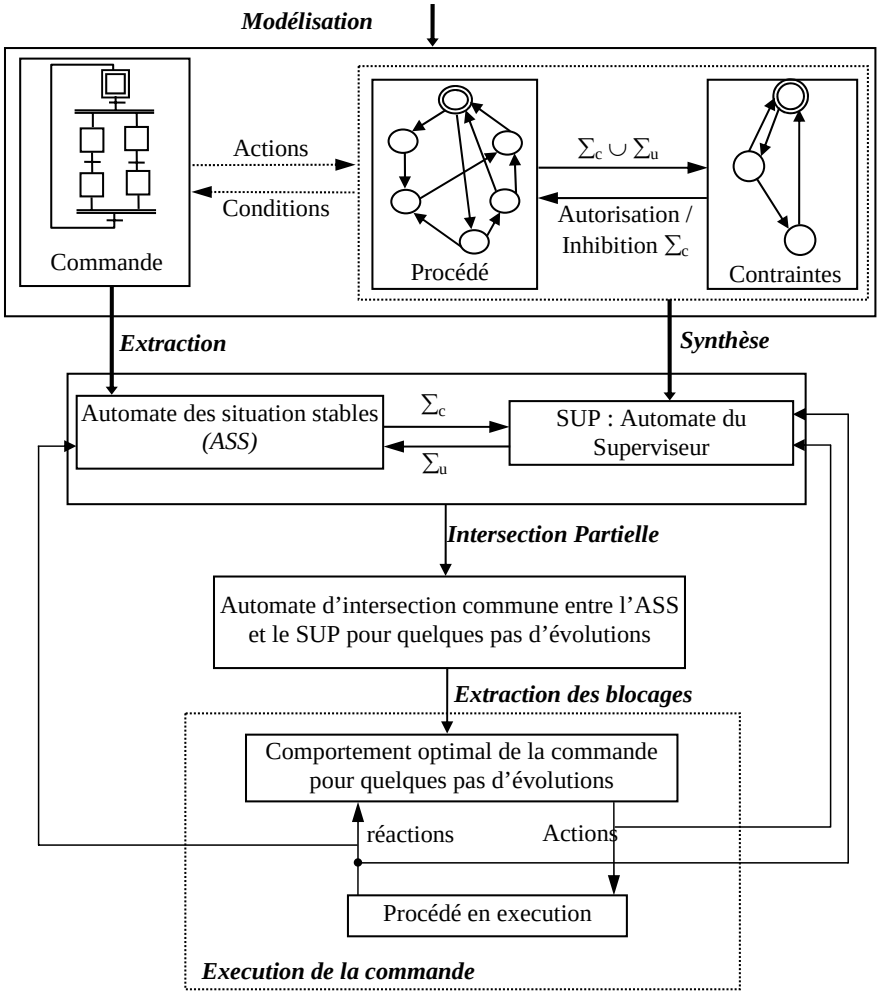


Figure 2.14 : Schéma du principe de la synthèse en ligne

5.2.2 Principe de l'intersection partielle

La procédure d'intersection partielle consiste à construire à partir des états courants de *ASS* et de *SUP*, le comportement qui leur est commun, pour quelques pas

d'évolutions futurs. Il en résulte un automate dénommé $SYNC^N$ où N représente le pas de calcul. N correspond au nombre de réactions consécutives du procédé prises en compte à partir de la situation courante du système.

Chacun des automates *ASS* et *SUP* repose sur une sémantique qui lui est propre. Pour *ASS*, un état est muni de l'ensemble des actions actives lors de la situation correspondante du Grafcet, et les transitions sont décrites par une fonction composée des entrées du Grafcet et de fronts sur ces entrées. Quant à *SUP*, il correspond à un automate événementiel. L'automate $SYNC^N$ se construit en tant que modèle asynchrone dont les transitions correspondent à des événements simples. Les ordres qui doivent être émis en parallèle dans une situation du Grafcet, sont représentés dans $SYNC^N$ par une structure arborescente de transitions décrivant les activations et les désactivations successives autorisées par le superviseur *SUP*. Les états reliés par ces transitions constituent une région correspondante à la dite situation du Grafcet. Par conséquent, pour une fenêtre d'intersection dont la taille est fixée par la valeur N, l'automate $SYNC^N$ est constitué d'un ensemble de régions, de transitions intra-régions correspondantes aux événements commandables et de transitions inter-régions aux événements non-commandables.

La construction de $SYNC^N$ s'effectue en temps réel sur une partie limitée de la commande, elle fournit un automate inachevé tel que celui de la figure 2.15. Ainsi, en ne s'intéressant qu'au nombre d'événements non commandables Σ_u, l'automate $SYNC^N$ est construit par rapport aux futures situations atteignables du Grafcet. Par conséquent, pour une région atteinte, toutes les actions associées à la situation Grafcet correspondante sont prises en compte. Les états de $SYNC^N$ n'ayant aucune transition sortante sont bloquants.

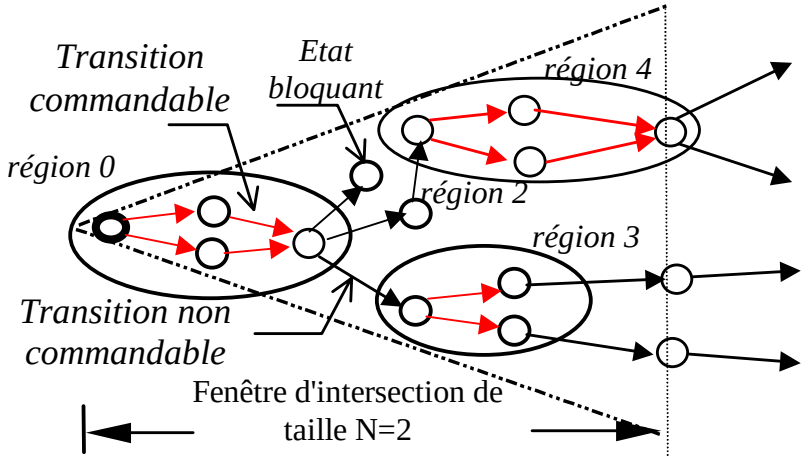


Figure 2.15. Modèle sémantique d'une fenêtre d'intersection partielle.

5.2.3 Formalisation de l'intersection partielle

L'automate $SYNC^N$ est défini par un 9-uplet (Σ, ST, TR, st_0, r_0, f_0, BLOC, EM, RD) avec :

- $\Sigma = \Sigma_c \cup \Sigma_u$, $\Sigma_c = \uparrow Z \cup \downarrow Z$ et $\Sigma_u = \uparrow E \cup \downarrow E$. Les ensembles d'événements $\uparrow Z$ et $\downarrow Z$, correspondent aux activations et désactivations des ordres, tandis que les ensembles d'événements $\uparrow E$ et $\downarrow E$ reflètent les fronts montant et descendant des entrées. Dans la suite, $\uparrow z$ correspond à un élément de $\uparrow Z$, $\downarrow z$ à un élément de $\downarrow Z$, $\uparrow e$ à un élément de $\uparrow E$ et $\downarrow e$ à un élément de $\downarrow E$.

- ST est l'ensemble des états de $SYNC^N$. ST est partitionné en sous-ensembles appelés régions. Un état $st \in ST$ correspond à une configuration unique du 3-uplet (q, y, E), où *q* est un état de *SUP*, *y* un état de *ASS* et *E* l'ensemble des valeurs logiques (0 ou 1) des entrées du Grafcet. Tous les états correspondants à une situation du Grafcet et à une configuration unique de ses variables d'entrée, sont groupés dans une région. Ces régions peuvent être groupées dans une fenêtre d'intersection partielle dont la taille dépend de *N*.

- $TR : ST \times \Sigma \rightarrow ST$ est une fonction partielle représentant les transitions de $SYNC^N$. Une transition est définie par un triplet $(st, \sigma, st') \in TR$.

- $st_0 \in$ ST est l'état initial qui est donné par (q_0, y_0, E_0), où q_0 est l'état initial de *SUP*, y_0 l'état initial de *ASS* et E_0 correspond à l'ensemble des valeurs initiales des entrées du Grafcet.

- $r_0 \subset$ ST est la région initiale correspondante à la situation initiale y_0 de *ASS* et aux valeurs initiales E_0 de ses variables d'entrée. Cette région inclut l'état initial st_0.

- BLOC $\subset$ ST contient l'ensemble des états bloquants, c'est-à-dire ceux n'ayant pas de transitions sortantes.

- EM $\subset$ ST contient l'ensemble des états et des régions marqués.

- RD $\subset$ ST contient l'ensemble des régions déjà développées.

Pour générer l'automate $SYNC^N$, l'étape d'intersection partielle est basée sur l'algorithme itératif donné en annexe 2, qui s'arrête lorsqu'aucune fenêtre d'intersection ne peut plus être créée.

5.2.4 Application de la synthèse en ligne

L'objectif est d'établir une comparaison entre la synthèse hors ligne et en ligne pour mettre en évidence le gain en espace mémoire ainsi que l'importance du choix de *N* qui définit la taille de la fenêtre d'intersection partielle. Nous présentons deux cas de figure : N=2 et N=10.

Pour cet exemple simple, l'étape de génération de la commande calculée hors ligne fournit un automate *SYNC* composé de 187 transitions et 95 états groupés dans 57 régions et un automate de commande *COR* composé de 57 états et 111 transitions. Pour une fenêtre de 2 pas, la première intersection partielle développée pour la situation initiale du système est donnée figure 2.16.a (automate $SYNC^2$). Au départ, l'état courant de la commande est l'état 0 et aucune action n'est à exécuter. Suite au départ cycle ($\uparrow dcy$), l'état 1 est activé et l'ordre *DESCENDRE* est envoyé. Le nombre d'événements non commandables courant *Ncou* est alors incrémenté. En parallèle, une nouvelle intersection partielle engendre l'automate de la figure 2.16.b qui débute par le nouvel état courant.

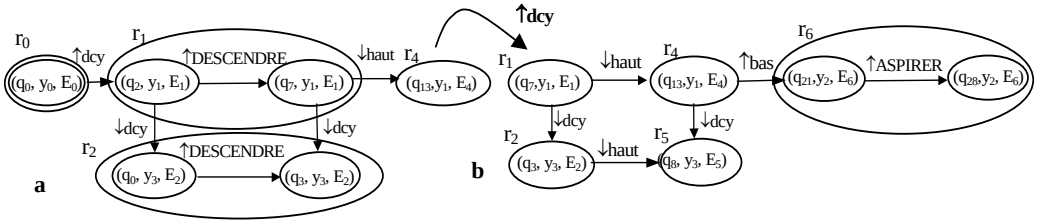


Figure. 2.16. Automates partiels d'intersection (*SYNC*^{2}).

Les contraintes imposées étant très restrictives, les automates représentant les intersections partielles sont de très petite taille. Pour chacune de ces intersections partielles, nous avons représenté la commande optimale (*COR*) correspondante en figure 2.17.a et 2.16.b. A chaque état est associé l'ensemble des ordres à envoyer (*ORD*) et à interdire par *COR* (*PROH)*. Ces ensembles sont utilisés pour renseigner l'utilisateur sur les ordres qui sont maintenus ou non par la procédure de synthèse. La région r_0 correspond désormais à l'état 0 de *COR* où aucun ordre n'est envoyé. Les deux états de la région r_1 sont agrégés en un unique état, l'état 1 où l'ordre envoyé pendant l'exécution est *DESCENDRE*, aucun ordre n'étant interdit.

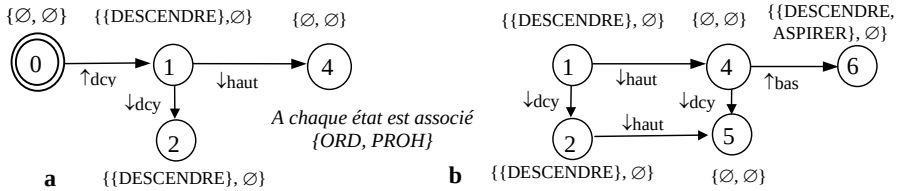


Figure. 2.17. Automates partiels de commande (*COR*).

Le tableau 2.2 compare les automates *SYNC* et *COR* obtenus par synthèse hors ligne et ceux obtenus en ligne pour N=2. Il montre le gain en espace mémoire pour le calcul de chaque intersection partielle.

	Synthèse hors ligne
SYNC	95 états, 187 transitions
OPT	57 états, 111 transitions

Synthèse en ligne	1ère intersection partielle	2ème intersection partielle	3ème intersection partielle	…
SYNC^2	6 états, 6 transitions	6 états, 6 transitions	11 états, 10 transitions	…
OPT^2	4 états, 3 transitions	5 états, 5 transitions	6 états, 6 transitions	…

Tableau 2.2 : Automates générés hors ligne et en ligne

La démarche de synthèse en ligne est maintenant appliquée pour N=10. A la différence du premier exemple, les intersections développées deviennent rapidement importantes en nombre d'états et de transitions. Ainsi, la première intersection partielle se compose de 31 états et 43 transitions dans 20 régions, ce qui donne un automate de commande de 20 états et 32 transitions. L'augmentation de la taille de la fenêtre

engendre donc une augmentation du nombre d'états et de transitions mais aussi une diminution du nombre d'intersections et donc un risque de réapparition des problèmes liés à la synthèse hors ligne. De plus, si notre exemple ne pose ici aucun problème, il se peut que pour une autre application la valeur N=10 soit conséquence de blocage. En effet, si une fenêtre de taille N contient une situation bloquante, et l'origine du blocage provient d'une situation précédente rencontrée à plus de N pas, alors l'origine du blocage n'est pas visualisable dans la fenêtre en cours, et donc non traitable. Le choix de la valeur N est donc un frein à l'utilisation de la synthèse en ligne et reste donc à approfondir.

6 DISCUSSIONS ET CONCLUSIONS

Nous avons présenté dans ce chapitre les travaux de base effectués dans le cadre d'une démarche de synthèse d'une commande sûre, réactive, déterministe et sans blocage décrite initialement par Grafcet. Nous avons ensuite illustré cette démarche autour d'un exemple issu du groupe de travail COSED du GDR MACS. La mise en œuvre de cette démarche fait apparaître plusieurs difficultés :

- La définition des modèles de PO et de contraintes reste une tâche très délicate, en particulier lorsque les systèmes à modéliser ont une grande taille. En effet, la modélisation des systèmes de complexité réelle demande une étude au préalable, basée sur l'utilisation de méthodes structurées plus ou moins formelles permettant de concevoir le fonctionnement global du système. Tous ceci nécessite le développement d'une méthodologie d'aide à l'élaboration d'un modèle de partie opérative conformément à la sémantique préconisée par notre approche et la définition du degré de granularité nécessaire pour les besoins du système traité.

- La prise en compte de modèles plus évolués pour la partie opérative et les contraintes va nécessiter soit de développer une extension de la théorie de supervision adaptée aux modèles utilisés ou de s'affranchir de cette théorie et de proposer une méthode autre.

- Le concepteur fournit actuellement des modèles et n'intervient plus dans la démarche de synthèse proprement dite qui est complètement automatique. Par conséquent, il serait souhaitable de procurer au concepteur une aide dans l'élaboration de la commande et dans le raffinement des modèles de départ, en lui permettant par exemple de visualiser et d'analyser les séquences bloquantes ainsi que les corrections proposées à travers les propositions de simulations partielles de ces modèles.

- Il faut disposer d'un module permettant d'identifier avant implantation, l'origine des blocages et les corrections qui peuvent être apportées au modèles de départ, soit pour le Grafcet ou soit pour la PO et les contraintes. Un tel module doit s'appuyer sur l'analyse de l'automate d'intersection, en fonction des situations Grafcet fixées par l'utilisateur.

La figure 2.18 illustre la nouvelle proposition de démarche. Elle porte sur la prise en compte de modèles plus évolués et structurés pour la partie opérative et les contraintes que les automates à états utilisés par la méthode de [NDJ 99] (figure 2.3). L'utilisation de ces nouveaux modèles va nécessiter de proposer une opération d'intersection adéquate entre le Grafcet et les nouveaux modèles. Cette opération d'intersection permet d'obtenir un automate SYNC représentant le comportement commun entre le Grafcet, la partie opérative et les contraintes. L'automate de commande final COR peut être obtenu soit par traitement automatique selon les principes évoqués précédemment (§ 2.2) soit par traitement semi automatique qui intègre le concepteur dans l'élaboration de la commande et dans le raffinement des modèles de départ. Le traitement semi automatique traite deux possibilités, la première correspond à la visualisation et à l'analyse des séquences bloquantes à travers une présentation de la trace des évolutions conduisant au blocage, la seconde consiste à appréhender une liste des contraintes non respectées et à en déduire une explication permettant au concepteur de revoir son Grafcet de commande. A l'issue des éventuels corrections du concepteur, une synthèse est relancée.

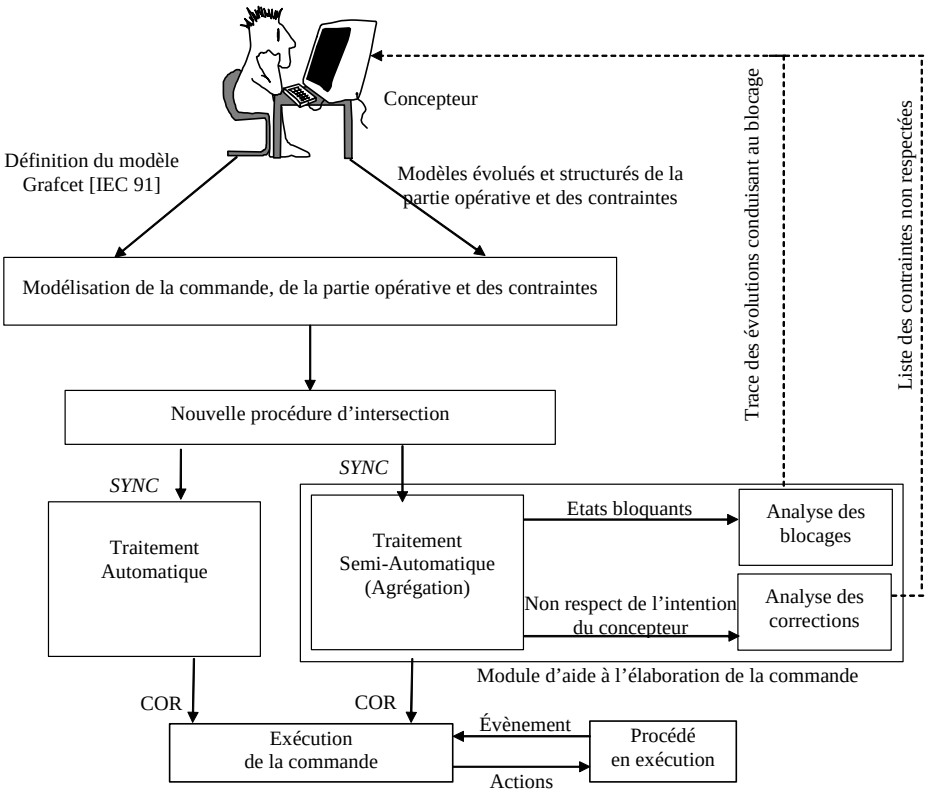


Figure 2.18. Nouvelle méthodologie de synthèse de commande.

Cette méthodologie de synthèse, va se construire à travers le chapitre 3 pour les modèles évolués et structurés proposés pour la partie opérative et les contraintes, et à travers le chapitre 4 pour la démarche de synthèse adaptée à ces nouveaux modèles ainsi que la synthèse sous contrôle du concepteur. La mise en œuvre de ces concepts sur un exemple de commande de tri de caisse est présentée dans le chapitre 5.

Chapitre 3

VERS LA PRISE EN COMPTE D'UNE MODELISATION PLUS EVOLUEE DE LA PARTIE OPERATIVE ET DES CONTRAINTES

L'une des étapes essentielles pour établir une démarche de synthèse de commande est la modélisation de la partie opérative (PO) et des contraintes. Cependant, une des grandes difficultés rencontrées dans l'élaboration de la synthèse réside dans la complexité de modélisation. Pour pallier cette difficulté et aider le concepteur dans la phase de modélisation, nous proposons dans ce chapitre des modèles évolués et structurés pour la PO et les contraintes. Pour la partie opérative, nous préconisons une structuration à base de règles afin d'obtenir un modèle automate à états et pour les contraintes, nous utilisons une modélisation par des équations logiques dans l'algèbre de Boole classique.

1 INTRODUCTION

La construction d'une commande correcte par synthèse automatique suppose l'élaboration des modèles pour la partie opérative et les contraintes de fonctionnement appelées aussi spécifications. L'utilisation de la théorie de supervision dans nos travaux antérieurs nous a contraint à modéliser les éléments de la partie opérative (PO) et les contraintes, par des modèles à base d'automates à état. Cependant, ce modèle manque de convivialité et d'expressivité et son élaboration est très intuitive. En effet, intégrer tous les détails d'un système dans un seul modèle, génère des difficultés liées d'une part, à l'explosion combinatoire et d'autre part, à des erreurs de modélisation.

Nous proposons dans ce chapitre de contribuer à résoudre cette problématique par une construction modulaire structurée. La prise en compte de ces différents aspects conduira à étendre la démarche de synthèse pour tenir compte de la sémantique des nouveaux modèles.

Ce chapitre est structuré en deux parties. La première partie présente une méthodologie de modélisation de partie la opérative à partir des règles de précédence, correspondant à l'influence des événements commandables sur les événements non commandables, et de relations de précédence représentant la chronologie entre les événements non commandables conséquents au même événement commandable❶. La deuxième partie présente une modélisation des contraintes par des équations logiques ❷ dans l'algèbre de Boole classique. Ces méthodologies sont prises en compte dans la phase de modélisation de la démarche globale présentée par la figure 3.1. Le préhenseur pneumatique décrit au chapitre 2, illustre les différentes méthodologies proposées.

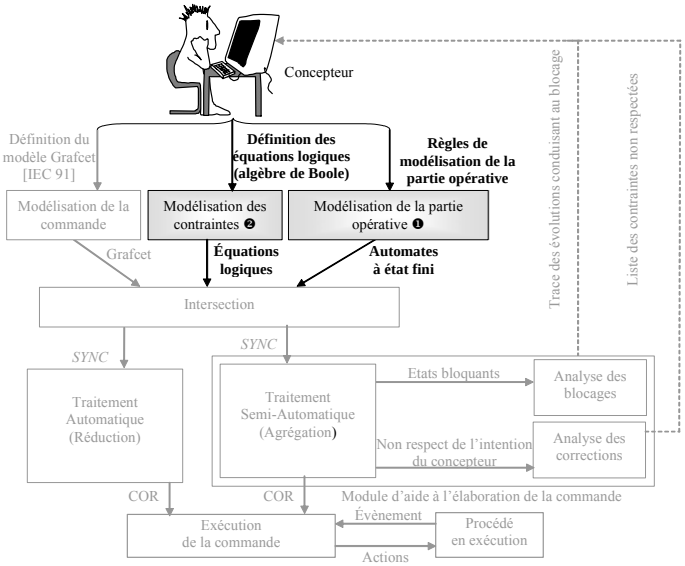


Figure 3.1. Méthodologie de synthèse de commande adoptée.

2 MODELISATION DE LA PARTIE OPERATIVE

2.1 Présentation

Les modèles intuitifs d'éléments de partie opérative que nous avons vus précédemment (chapitre 2), sont complexes et conduisent à se poser plusieurs questions : Comment vérifier que rien n'a été oublié en termes d'états et de transitions, comment intégrer de nouveaux éléments dans ces modèles sans tout recomposer ? La modélisation proposée actuellement fait appel non seulement à une bonne connaissance du système, mais aussi à l'expérience, et aux compétences en terme de modélisation du concepteur.

D'une part, lors de l'élaboration des modèles du système, le concepteur a tendance à faire l'amalgame entre le modèle de la commande, le modèle de la partie opérative et le modèle des contraintes. Par exemple, au niveau de la commande, le concepteur cherche implicitement à intégrer une image de la partie opérative dans le Grafcet afin d'exprimer la causalité entre l'envoi des ordres et les réactions conséquentes de la partie opérative due à la modélisation intuitive. Ceci suppose cependant, que toutes les réactions du procédé et leur ordonnancement, soient connus de manière précise pour chaque action et chaque situation de la commande. C'est illusoire, car la complexité et la nature parallèle des systèmes de commande et des procédés à commander entraînent des évolutions parallèles et complexes entre l'envoi des ordres et les réactions conséquentes du procédé.

D'autre part, le concepteur a tendance à intégrer implicitement les contraintes de sécurité afin d'exprimer au plus tôt, des restrictions sur le comportement du procédé. Dans l'exemple du préhenseur présenté au chapitre 2, le modèle du mouvement horizontal intègre la contrainte de sécurité « Ne pas avancer et reculer en même temps ». La contrainte est dans cet exemple, simple à exprimer car il s'agit d'un modèle élémentaire de partie opérative mais ceci risque d'être beaucoup plus complexe lorsqu'il s'agit de modèles où la granularité du comportement modélisé est différente. D'autre part, quand des éléments du système doivent être ajoutés, retirés ou configurés différemment, le concepteur doit revoir l'ensemble des modèles et leurs interactions.

Il est donc préférable de s'appuyer sur des modèles explicites et indépendants entre eux pour représenter la commande, la partie opérative et les contraintes. Néanmoins, la modélisation explicite de la partie opérative est une tâche complexe qui se heurte non seulement à des problèmes méthodologiques de structuration et de composition de ses

éléments mais aussi au choix de la granularité du comportement modélisé. Par conséquent, nous proposons dans la suite une méthode structurée à base de règles en s'appuyant sur des travaux de V. Chandra [CHA 01][CHA 02], pour aider le concepteur dans sa tâche de conception des modèles de partie opérative.

2.2 Modélisation structurée de la partie opérative à base de règles

2.2.1 Règles d'occurrence et relations de précédence

Pour structurer la modélisation de la partie opérative, nous avons retenu une modélisation à base de règles définissant les interactions entre les événements commandables et les événements non commandables, et des relations entre ces événements non commandables. Il s'agit dans le premier cas de règles dites d'occurrence et dans le second cas, de relations de précédence. Ces règles définies par l'utilisateur sont ensuite traduites en automates à états. Nous précisons, que cette analyse à base de règles d'occurrence et de relations de précédence, est appliquée à un modèle fonctionnel.

Soit $\sum_c$ l'ensemble des événements commandables, $\sum_c = \uparrow Z \cup \downarrow Z$ tel que les ensembles $\uparrow Z$ et $\downarrow Z$, correspondent aux activations et désactivations des ordres. Avant de déterminer les règles d'occurrence des événements commandables $\sum_c$, il faut commencer par fixer le contexte c'est-à-dire les conditions initiales CI du système. Il s'agit ensuite de recenser tous les événements liés à l'élément de partie opérative que l'on cherche à modéliser puis de définir sous forme de règles, l'influence de l'activation et de la désactivation des événements commandables $\sum_c$ sur les événements non commandables $\sum_u = \uparrow E \cup \downarrow E$. Par conséquent, chaque règle va s'exprimer selon le principe simple d'une « cause/conséquence », la cause est liée à l'événement commandable $\downarrow z$ ou $\uparrow z$ tel que $z \in Z$ et la conséquence porte sur l'événement non commandable $\uparrow e$ ou $\downarrow e$ tel que $e \in E$. Pour chaque règle d'occurrence ayant la même cause, il faut ensuite établir sous forme de relations de précédence, la chronologie liant les événements non commandables conséquents.

Formellement, le principe est le suivant :

- Pour les systèmes à événements discrets, nous définissons souvent dans la phase de modélisation de la partie opérative les conditions initiales des entrées (capteurs) et des sorties (Actionneurs) du système. Ces états initiaux correspondent soit aux activations, soit aux désactivations des entrées/ sorties du système. Autrement dit, ce sont les valeurs initiales des événements commandables $\sigma \in \sum_c$ (Action) et non commandables $\alpha \in \sum_u$ (Réaction) associées

au système.

•L'interaction entre les sorties (actionneurs) et les entrées (capteurs) permet de déterminer des règles d'occurrence de type cause/conséquence. Il s'agit en fait de déterminer s'il existe des événements non commandables α du système qui sont générés suite à l'occurrence d'un événement commandable σ. Formellement, l'écriture de cette proposition est la suivante :

$\forall\sigma\in\sum_c$ avec $\sum_c = \uparrow Z \cup \downarrow Z$ et si $\exists\alpha\in\sum_u$ avec $\sum_u = \uparrow E \cup \downarrow E$ tel que α est une conséquence de σ alors on définit la règle d'occurrence :

$$\sigma \to \alpha \text{ avec } \begin{cases} \sigma = \uparrow z \ ou \downarrow z \\ \alpha = \uparrow e \ ou \downarrow e \end{cases}$$

• Un événement commandable peut conduire à générer un ou plusieurs événements non commandables. Pour séparer ces événements non commandables en terme de précédence nous définissons pour chaque règle d'occurrence ayant la même cause, une relation appelée « relation de précédence », permettant de préciser la chronologie liant les événements non commandables conséquents, telle que :

$\forall\sigma\in\sum_c$ si $\exists\ \alpha_1, \alpha_2 \in \sum_u \times \sum_u$ et $\sigma \to \alpha_1, \sigma \to \alpha_2$ alors on définit une relation de précédence entre α_1 et α_2 par :

$$\alpha_1 \to \alpha_2 \text{ ou } \alpha_2 \to \alpha_1.$$

A partir de ces règles d'occurrence et de ces relations de précédences est dérivé automatiquement un modèle automate à état équivalent du système. Dans cet objectif, nous proposons deux méthodes de construction automatique ; une première construction avec une composition d'automate et une deuxième avec une construction booléenne.

2.2.2 Construction de l'automate équivalent de PO par composition synchrone d'automate d'occurrence et de précédence

a. Méthode

La construction de l'automate de la partie opérative s'appuie sur les informations précédemment acquises (les conditions initiales, les règles d'occurrence et les relations de précédences) et s'effectue selon quatre étapes:

❶ La première étape consiste à construire l'automate d'occurrence G_{oc} à 2^n états (n étant le nombre de sorties de la commande) décrivant toutes les évolutions possibles des événements commandables à partir de l'état initial donné.

L'automate G_{oc} est défini par $G_{oc} = (Q_{oc}, \sum_{oc}, T_{oc}, \delta_{oc}, q_0)$, avec Q_{oc} est l'ensemble des états de l'automate de cardinal n (Card (Q_{oc}) = n), $\sum_{oc} = \sum_{occ} \cup \sum_{ocu}$ est défini par l'union de l'ensemble des événements commandables $\sum_{occ}$ et de l'ensemble des événements non commandables, T_{oc} est l'ensemble de transitions de l'automate G_{oc}, δ est la fonction de transition définie par δ_{oc} $(q, \sigma)=q'$, enfin q_0 est l'état initial. La structure initiale de l'automate G_{oc} est $G_{oc0}= \{\{q_{0oc}\}, \varnothing, \varnothing, \delta_{oc}, q_{0oc}\}$. L'algorithme (figure 3.2) de construction de cet automate consiste à ajouter tous les événements commandables $\sum_c$ recensés par les règles à l'ensemble des événements $\sum_{oc}$, à ajouter les états atteignables à l'ensemble Q suite à l'occurrence des événements que l'on vient de recenser, et les transitions correspondantes à T_{oc}.

1- $\sum_{oc}= \sum_{oc} \cup \sum_c$.

2- $\forall \sigma \in \sum_{oc}$, si $q_{oc}' = \delta(q_{oc}, \sigma)$

$Q = Q \cup \{q_{oc}'\}$ si $q_{oc}' \notin Q_{oc}$

$T_{oc} = T_{oc} \cup \{(q_{oc}, \sigma, q_{oc}')\}$

Figure 3.2. Construction de l'automate d'occurrence.

❷ La deuxième étape consiste à compléter l'automate obtenu précédemment, par des transitions rebouclantes sur les états de l'automate d'occurrence (figure 3.3). Ces transitions représentent les événements non commandables résultant des règles d'occurrence. Leur présence dans un état indique que les événements associés sont autorisés. Dans certains états de l'automate, une entrée peut en même temps être autorisée en activation ou en désactivation.

$\forall \sigma \in \sum_{occ}$ et $\forall \alpha \in \sum_u$ tel que $\sigma \rightarrow \alpha$ alors

$\forall (q',q) \in Q_{oc}^2$ tel que $\delta(q', \sigma)=q$. alors

$T_{oc}= T_{oc} \cup \{(q, \alpha, q)\}$ et $\sum_{oc}= \sum_{oc} \cup \{\alpha\}$.

Figure 3.3. Complément avec les évolutions non commandables.

❸ La troisième étape concerne la construction d'un automate dit de précédence G_{pred} à partir des relations de précédence entre les événements non commandables et de la situation initiale.

❹ La quatrième étape consiste à calculer le produit croisé synchrone entre l'automate G_{oc} issu des règles d'occurrence et de l'automate de précédence G_{pred} jusqu'à obtenir

l'automate final G défini par G = {Q, Σ, δ, q_0}.

b. Exemple

Pour illustrer la méthode structurée de modélisation pour la partie opérative, nous reprenons ici l'exemple du préhenseur.

b.1. Définition des règles d'occurrence et de précédence

Pour le mouvement horizontal défini par un vérin double effet (Figure 3.4), il existe quatre événements commandables *↑(↓)AVANCER* et *↑(↓)RECULER*, et quatre événements non commandables *↑(↓)gauche* et *↑(↓)droite*. On prend l'hypothèse suivante : en situation initiale, le vérin se trouve à gauche et aucun ordre n'est envoyé à la partie opérative. Afin de définir les règles d'occurrence, il faut s'intéresser aux conséquences des ordres envoyés. L'activation de l'ordre *AVANCER* permet au vérin de se déplacer vers la droite et de quitter la position *gauche*. De même, l'activation de *RECULER* fait revenir le vérin de la position *droite* jusqu'au fin de course gauche. Suite à l'activation d'*AVANCER*, la chronologie des événements non commandables est la suivante : désactivation de *gauche* puis activation de *droite*. Pour la relation de précédence liée à l'activation de l'ordre *RECULER*, la chronologie se présente ainsi : désactivation de *droite* avant activation de *gauche*. Les conditions initiales et les différentes relations concernant le mouvement horizontal sont résumées dans le tableau 3.1.

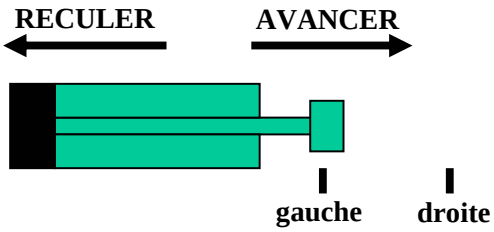


Figure 3.4. Vérin pneumatique: Mouvement Horizontal.

Eléments de la PO	Conditions Initiales	Règles d'occurrence	Relations de précédence
AVANCER *RECULER* *droite* *gauche*	$\overline{AVANCER}$ $\overline{RECULER}$ $\overline{droite}$ *gauche*	*↑AVANCER → ↓gauche* *↑AVANCER → ↑droite*	*↓gauche précède ↑droite*
		↑RECULER → ↓droite *↑RECULER → ↑gauche*	*↓droite précède ↑gauche*
		↓AVANCER → ↑droite	*néant*
		↓RECULER → ↑gauche	*néant*

Tableau 3.1 : Paramètres du modèle de mouvement horizontal.

Pour les autres sous modules de la partie opérative de l'exemple, nous procédons de la

même façon pour obtenir les règles d'occurrence et de précédence présentées dans le tableau 3.2.

Elément de la PO	Conditions Initiales	Règles d'occurrence	Relations de précédence
DESCENDRE *bas* *haut*	$\overline{DESCENDRE}$ $\overline{bas}$ $\overline{haut}$	$\uparrow DESCENDRE \rightarrow \downarrow haut$ $\uparrow DESCENDRE \rightarrow \uparrow bas$	$\downarrow haut$ *précède* $\uparrow bas$
		$\downarrow DESCENDRE \rightarrow \uparrow haut$ $\downarrow DESCENDRE \rightarrow \downarrow bas$	$\downarrow bas$ *précède* $\uparrow haut$

a. Le mouvement vertical.

Eléments de la PO	Conditions Initiales	Règles d'occurrence	Relations de précédence
ASPIRER *pièce*	$\overline{ASPIRER}$ $\overline{pièce}$	$\uparrow ASPIRER \rightarrow \uparrow pièce$ $\downarrow ASPIRER \rightarrow \downarrow pièce$	$\uparrow pièce$ *précède* $\downarrow pièce$

b. Aspiration.

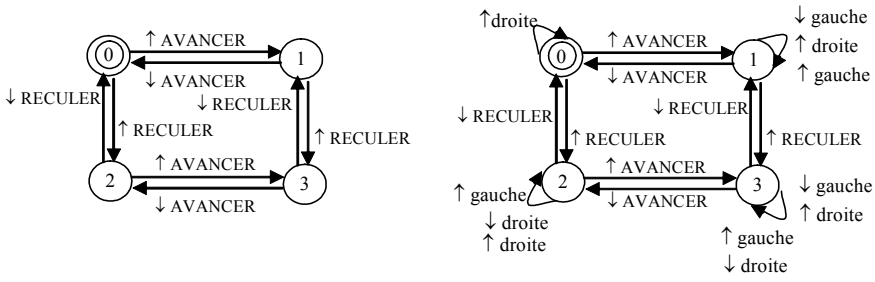
Tableau 3.2 : récapitulatifs des règles pour le mouvement vertical et l'aspiration.

b.2. Construction de l'automate équivalent

Le modèle élaboré est constitué de trois automates résultants décrivant respectivement le mouvement horizontal (Figure.3.7), le mouvement vertical (Figure 3.8a), le mouvement d'aspiration (Figure 3.8b.).

Les étapes ❶ et ❷ de cette méthode consistent donc à établir pour le mouvement horizontal un automate à 4 états avec des évolutions entre états liées à l'activation ou la désactivation de l'ordre *AVANCER* et de l'ordre *RECULER*, puis compléter cet automate par les événements non commandables gauche et droite en respectant les règles d'occurrence définies par l'utilisateur. Ces étapes sont illustrées dans les figures 3.5a et 3.5b.

Aucun ordre n'étant activé dans les conditions initiales, l'état 0 de la figure 3.5a permet soit l'activation de l'ordre *AVANCER* (état 1), soit l'activation de l'ordre *RECULER* (état 2). A partir de l'état 1, la désactivation de *AVANCER* est alors possible, ce qui permet de retourner à l'état initial. A partir de l'état 1, l'activation de l'ordre *RECULER* est également possible et permet de se rendre à l'état 3. De même, à partir de l'état 2, il est possible de retourner à l'état 0 par désactivation de *RECULER* ou de partir vers l'état 3 par activation de *AVANCER*.



a) Automate des événements commandables

b) Automate avec la prise en compte des événements non commandables

Figure 3.5. 1er et 2ème pas de construction de l'automate G_{oc}

A partir de cet automate, il faut ajouter des boucles sur les états en prenant en compte les événements non commandables définis dans les règles d'occurrences (Figure 3.5b). Suite à l'activation de l'ordre *AVANCER* (états 1 et 3 de l'automate précédent), il est possible de désactiver *gauche* et d'activer *droite*. L'activation de l'ordre *RECULER* entraîne quant à lui des boucles dans les états 2 et 3 avec l'activation de *gauche* et la désactivation de *droite* comme événements non commandables.

L'étape ❸ conduit à établir avec les deux relations de précédence (tableau 3.1) liant *gauche* et *droite* (tableau 3.1), l'automate de précédence illustré dans la figure 3.6. A l'initialisation, le vérin est à gauche (état 0), l'activation de droite (état 2) ne peut se faire qu'après une désactivation de gauche en état 1. De même, pour retourner en état 0, la désactivation de droite (état 1) précède l'activation de gauche. Chaque état de cet automate autorise bien sûr tous les événements Σ_c du système.

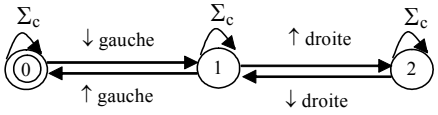


Figure 3.6. Automate de précédence G_{prec}

Pour obtenir l'automate final du mouvement horizontal (Figure 3.7), il suffit alors d'effectuer un produit croisé synchrone (étape ❹) entre l'automate issu de l'étape 2 et l'automate de précédence issu de l'étape 3. Le mouvement est alors décrit de façon complète indépendamment de la commande à travers 12 états et 36 transitions.

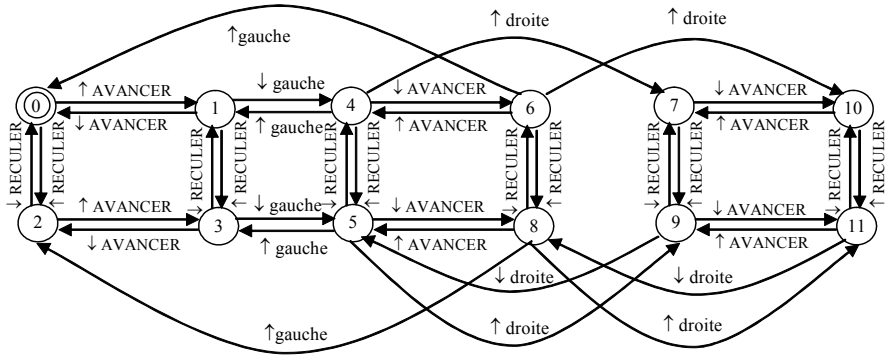
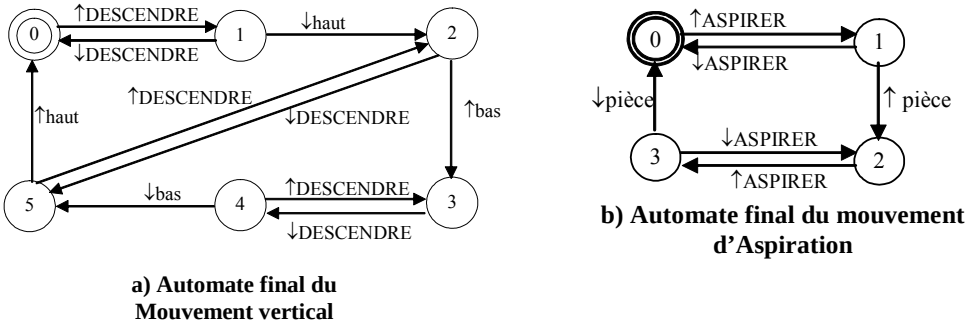


Figure 3.7. Automate final du mouvement horizontal

La construction des automates du mouvement vertical (figure 3.8a) et du mouvement d'aspiration (figure 3.8b), s'effectue de manière identique. Dans un objectif de synthèse et vu que notre démarche reste centralisée, nous allons effectuer un produit croisé asynchrone entre les différents éléments de la PO pour obtenir un automate global de 576 états et 3744 transitions.



a) Automate final du Mouvement vertical

b) Automate final du mouvement d'Aspiration

Figure 3.8. Modélisation structurée des différents sous systèmes composant le préhenseur

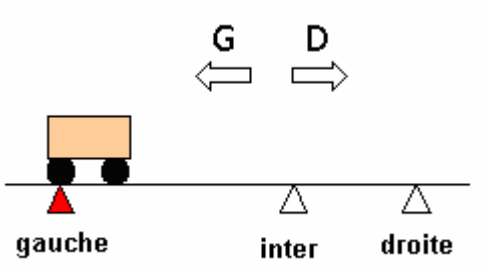
Discussion

Une comparaison avec le modèle intuitif de la figure 2.9a du chapitre 2, du mouvement horizontal montre clairement que le nouveau modèle (figure 3.5) est indépendant des contraintes spécifiées. En effet, dans l'état 3 de l'automate équivalent (figure 3.7), les deux ordres *AVANCER* et *RECULER* sont envoyés en même temps à la partie opérative, et ce sera la prise en compte de la contrainte qui viendra interdire cette situation. Cette méthodologie qui n'est pas restrictive et qui est indépendante des modèles de spécifications, génère par conséquent un nombre d'états et de transitions plus important que celui du modèle intuitif (432 états et 2664 transitions). Cependant, cette méthodologie généré un problème important lié à la construction de l'automate de précédence et qui concernent l'indéterminisme.

Par exemple, dans le cas de déplacement d'un chariot de gauche à droite avec trois capteurs *gauche*, *inter* (capteur de la position intermédiaire) et *droite*, les règles d'occurrence sont définies par :

↑*D* : ↓*gauche* puis ↑*inter* puis ↓*inter* puis ↑*droite.*

↑*G* : ↓*droite* puis ↑*inter* puis ↓*inter* puis ↑*gauche.*



L'automate de précédence équivalent est présenté dans la figure 3.9. Dans cet automate, nous constatons qu'à partir de l'état 2 et suite à l'occurrence de l'événement ↓*inter* les deux états 1 et 3 sont atteignables en même temps d'où le problème d'indéterminisme.

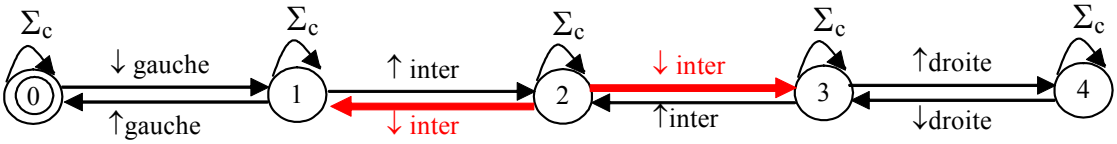


Figure 3.9. Automate de précédence.

Pour résoudre cette problématique, une deuxième méthode de construction à base toujours de règles est proposée. Il s'agit d'une méthode de construction par modélisation booléenne.

2.2.3 Méthode de construction par modélisation booléenne

a. Méthode

Dans cette partie, nous présentons une adaptation de la modélisation de la partie opérative basée sur une structure booléenne [WAN 00].

La structure des automates à état rudimentaire adoptée pour les SED et introduite par la théorie de supervision [RAM 89] n'est parfois pas assez explicite car les états n'apportent pas toutes les informations sur le système, spécialement quand le nombre d'états est assez grand. Les systèmes à événements discrets ont une structure logique liée à la nature booléenne des éléments de la partie opérative (par exemple, le capteur gauche peut avoir seulement l'une des deux valeurs 0 ou 1), il sera donc plus aisé, d'exploiter cette structure en utilisant des modèles d'automate à états booléens. [WAN 00].

a.1. Cadre formel de modélisation

Définition

Un automate à état booléen est défini par un 4-uplet G = (P, $\sum$, δ, 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 avoir la valeur 0 ou 1. Le nombre d'états maximal du modèle automate est décidé par le nombre des variables d'états, c'est-à-dire pour n variables on a 2^n états.

Comme défini dans la théorie des automates [RAM 89] [WON 02], δ est une fonction de transition $p_2 = \delta(p_1,\sigma)$, avec p_2, $p_1 \in P$. Dans le cas des automates à états booléens, on a $P \subseteq B^n$ et σ est un évènement de $\sum$. Soit p_{2i}, p_{1i} les $i^{ème}$ composantes respectivement de p_2 et de p_1. L'occurrence d'un événement σ va permettre de changer la $i^{ème}$ variable d'état correspondant ce qui peut être traduit par : Si $p_2 = \delta(p_1,\sigma)$ alors $p_{2i} = \neg p_{1i}$ et $p_{2k} = p_{1k}$ avec $k \neq i$.

Définition

Soit le vecteur d'entrées E, le vecteur de sorties Z et le label x associés à un état p de l'automate G tel que $E = [e_1,\ldots, e_m]$, $Z = [z_1,\ldots, z_n]$ et $x=\{x_1,\ldots, x_l\}$ où m est le nombre d'entrées du système, n le nombre de sorties du système et l le nombre d'états possibles dans l'automate G. L'ensemble des états de G est défini par :

$$P = \{[[z_1,\ldots, z_n], [e_1,\ldots, em], x_k] / z_i, e_i \in \{0,1\}^*\{0,1\}, j = 1,\ldots, n ; i = 1,\ldots, m$$
$$\text{et } k =1,\ldots,l\} \subseteq IB^n.$$

Ici, nous préconisons, la correspondance suivante : l'activation d'un événement commandable correspond à une mise à 1 (**S**et ou **S**) de la composante correspondante dans le vecteur Z, par contre la désactivation d'un événement commandable correspond à sa mise à 0 (**R**eset ou **R**). De la même manière, les événements non commandables sont traités par la mise à jour du vecteur d'entrée E.

De façon plus explicite, nous écrivons la structure de transition comme suit :

- L'activation d'une entrée *e* ou d'une sortie *z* est décrite par une mise à 1 donc,

  $S_e = 1$ ($S_z = 1$) et $R_e = 0$ ($R_z = 0$).

- La désactivation d'une entrée *e* ou d'une sortie *z* est définie par

$S_e = 0$ ($S_z = 0$) et $R_e = 1$($R_z = 1$).

- Si on n'a pas de changement d'événements alors on a S=0 et R=0 donc une mémorisation de l'état précédent, il n'y a pas de changement d'état.
- Le cas où S=1 et R=1 n'a aucune signification dans la modélisation que nous adoptons, il est impossible, on a soit mise à 1(**Set**) soit mise à 0(**Reset**).

L'ensemble de ce fonctionnement correspond ainsi au fonctionnement d'une bascule asynchrone RS

Soit donc, p_k un état du procédé G défini par le triplet (x_k, E_k, Z_k) avec E_k= [e_{k1}, …, e_{km}], Z_k= [z_{k1}, …, z_{kn}] et k=1,...,l l étant le nombre d'états possibles. Suite à l'occurrence d'un événement σ_k, l'état atteignable p_{k+1} est déterminé par :

$p_{k+1} = \delta(p_k, \sigma_k)$ avec $p_{k+1}= (E_{k+1}, Z_{k+1}, x_{k+1})$, $E_{k+1}= \delta(E_k, \sigma_k)$, $Z_{k+1}= \delta(Z_k, \sigma_k)$ et $x_{k+1} = \delta(x_k, \sigma_k)$.

$$e_{k+1,i}= RS(e_{k,i}) \text{ avec } i=1,..,m$$

$$z_{k+1,j}= RS(z_{k,j}). \text{ avec } j=1,..,n$$

avec $E_{k+1}= [e_{(k+1)1}, \ldots, e_{(k+1)m}]$ et $Z_{k+1}= [z_{(k+1)1}, \ldots, z_{(k+1)n}]$.

a.2. Méthode de construction

Après la définition des règles d'occurrence et des relations de précédence, nous présentons ici une méthode booléenne qui consiste à définir les combinaisons booléennes possibles représentant le vecteur des entrées/sorties du système et les incohérences logiques entre les entrées. Cette structuration définie par le concepteur conduit à construire un automate à état compatible avec la notation et la sémantique retenue dans la démarche de synthèse.

Les étapes nécessaires pour obtenir cet automate sont :

❶ Construction des combinaisons logiques possibles

Cette étape consiste à construire les 2^n états du système (n étant le nombre d'entrées/sorties) correspondant à 2^n combinaisons possibles des entrées/sorties du système. Cet ensemble correspond à l'ensemble des états Q.

Exemple : soit Z le vecteur de sorties du système et E le vecteur d'entrées avec Z=[z_1, z_2] et E=[e_1, e_2]. Donc, le nombre de combinaisons possibles est $2^4 = 16$ états correspondants à l'ensemble des états possibles.

états	z_1	z_2	e_1	e_2
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0

❷ Définir les incohérences logiques liées aux capteurs (entrées) du système.

Ces incohérences logiques sont déterminées par le cahier des charges exprimant ainsi des situations impossibles à atteindre dans un comportement normal de la partie opérative. Par exemple, dans le cas d'un mouvement vertical effectué au niveau du préhenseur, on ne peut pas avoir *haut* et *bas* en même temps. Après la définition de ces incohérences, les états correspondants q_{inc} sont supprimés de l'ensemble des états Q tel que :

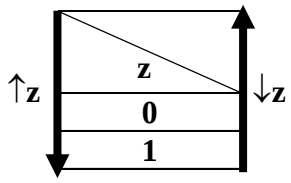
$$Q = Q \setminus \{q_{inc}\}$$

Exemple : En spécifiant l'incohérence logique $e_1=e_2=1$, on a $Q = Q\setminus\{3, 7,11, 15\}$.

états	z_1	z_2	e_1	e_2	
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	

❸ Construction de l'automate des évolutions commandables.

L'automate des évolutions commandables est déterminé à partir de la table de vérité exprimant toutes les évolutions possibles entre les états de *Q*. Ces évolutions sont définies seulement par des événements commandables $\uparrow z$ et/ou $\downarrow z$ de Σ_c. Sachant que chaque élément z du vecteur Z, peut prendre la valeur 1 dans le cas de l'activation de cette sortie *($\uparrow z$)* et 0 dans le cas de sa désactivation *($\downarrow z$)*.

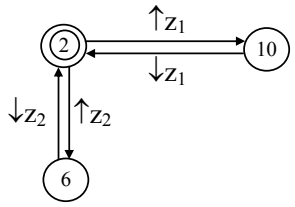


Le principe consiste à construire à partir d'un état q_k de Q (q_k = (x_k, E_k, Z_k) avec Z_k = [z_{1k},……., z_{nk}] et E_k= [e_{1k},……., e_{mk}]) l'état atteignable q_{k+1} suite à l'occurrence

d'un événement commandable $\uparrow z_{ik}$ ou $\downarrow z_{ik}$.

Exemple :

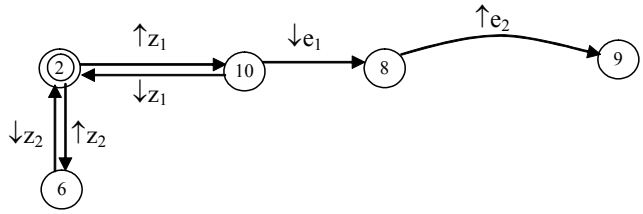
Dans la table de vérité définie dans la première étape, les deux éléments z_1 et z_2 de l'état (2) sont nuls. Dans ce cas, seule l'activation de z_1 ou de z_2 est possible pour atteindre l'état (10) ou l'état (6).



❹ Compléter l'automate avec les évolutions non commandables. Cette opération consiste à utiliser les règles d'occurrence et les relations de précédence.

Pour chaque état q_k de Q, les événements commandables qui y conduisent permettent de déterminer les états atteignables q_{k+1}. La règle d'occurrence permet de déterminer l'état atteignable dans l'ensemble Q. Cependant, la règle d'occurrence est souvent composée de plusieurs causes/conséquences, d'où l'intérêt de la relation de précédence entre ces conséquences. La relation de précédence entre les événements non commandables permet de définir la séquence d'évolution depuis q_k (q_k, q_{k+1}, q_{k+2}).

Exemple : supposons que dans la règle d'occurrence liée à z_1 on a $\uparrow z_1 \rightarrow \downarrow e_1$ et $\uparrow z_1 \rightarrow \uparrow e_2$ et $\downarrow e_1$ précède $\uparrow e_2$. L'automate résultant de la 4ème étape est le suivant.



b. Illustration

Dans cette partie, nous reprenons l'exemple du mouvement horizontal.

❶ Construction des différents états (combinaisons) du système :

états	*AVANCER*	*RECULER*	*gauche*	*droite*	
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

Tableau 3.3. Etat du système (incohérences).

❷ Supprimer les combinaisons logiques de l'ensemble des états représentant les incohérences logiques entre les entrées (Capteurs). Ici, c'est *gauche* = *droite* = 1.

❸ Construire les transitions entre les états de la table de vérité.

gauche/droite / **AVANCER/RECULER**	**00**	**01**	**11**	**10**
00	0	1		2
01	4	5		6
11	12	13		14
10	8	9		10

Incohérence logique

Diagramme des évolutions commandables.

Cette construction consiste à définir les transitions commandables sortantes de chaque état du diagramme ainsi que leurs états atteignables.

A partir de l'état initial **(2)** correspondant à la combinaison [0010] les événements commandables possibles, sont *↑AVANCER* et *↑RECULER*. Les états atteignables depuis cet état sont donc les états (10) et (6) (figure 3.10). Les autres évolutions commandables entre les états restants sont définies de la même manière.

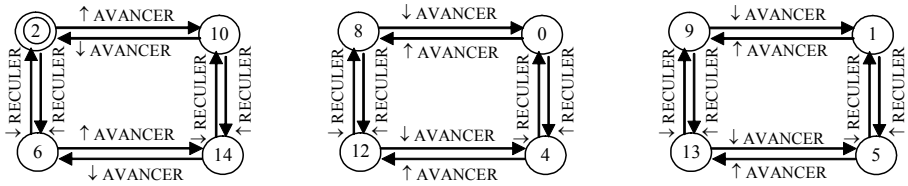


Figure 3.10. Construction de l'automate des transitions commandables.

❹ Construction des évolutions non commandables

L'état initial (2) est atteint après occurrence des deux événements *↓AVANCER* et *↓RECULER*. Pour l'événement *↓AVANCER,* la règle d'occurrence associée est définie par la conséquence *↑droite*. Or, cette évolution est impossible, car le système se trouve à gauche, c'est une incohérence logique. Pour *↓RECULER*, la conséquence définie est *↓gauche*, or le système se trouve déjà dans cette position. Par conséquent aucune évolution non commandable n'est possible à partir de cet état.

L'état (10) atteint après occurrence des deux événements *↓RECULER* et *↑AVANCER*. Pour l'événement *↓RECULER*, aucune évolution non commandable n'est possible car la règle d'occurrence associée est définie par la conséquence *↑gauche* et le système se trouve déjà à gauche, par contre, pour *↑AVANCER*, la règle d'occurrence et la relation de précédence est définit par l'occurrence de *↓gauche* puis *↑droite*. Ceci permet alors d'atteindre l'état (8) puis l'état (9).

Les autres transitions sont ensuite ajoutées de la même manière. L'automate final obtenu est présenté dans la figure 3.11. Cet automate comporte 12 états et 36 transitions.

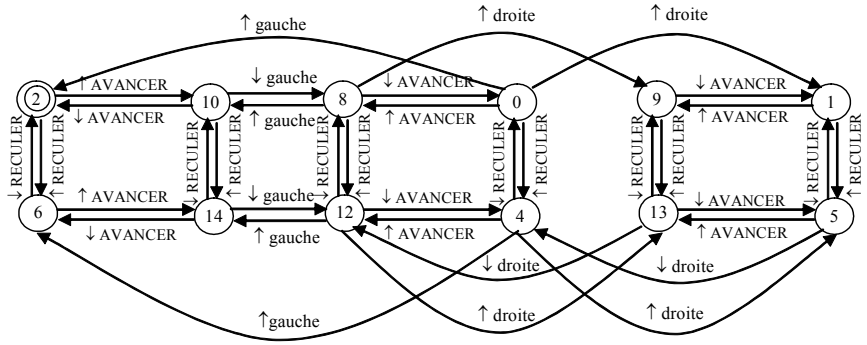


Figure 3.11. Modèle obtenu après ajout des différentes transitions non commandables (automate final du mouvement horizontal).

Cette structuration booléenne permet alors d'éliminer l'intuitivité et l'indéterminisme imposé par la construction par automate de précédence et d'éviter le produit croisé qui

complexifie le calcul de l'automate de partie opérative.

Pour illustrer ces propos, nous reprenons le même exemple du chariot présenté précédemment. Un extrait de l'automate à états booléens obtenu par la méthode de construction booléenne est présenté dans la figure 3.12. Nous constatons que le problème d'indéterminisme lié à l'événement non commandable *inter* rencontré dans la 1ère méthode est levé. En effet, l'occurrence de l'événement ↑*inter* conduit vers l'état 18 et sa désactivation ↓*inter* conduit vers l'état 16 pour continuer après son déplacement vers *droite* (état 17). Par contre, dans le cas du retour du chariot de droite à gauche, l'occurrence de l'événement ↑*inter* à l'état 8 conduit le système à l'état 10 puis sa désactivation de l'état 10 vers l'état 8. Nous constatons donc que, dans cet exemple, le problème d'indéterminisme est supprimé. En effet, l'occurrence d'un événement permet le passage d'un état à un autre et non pas dans deux états différents, levant ainsi l'indéterminisme.

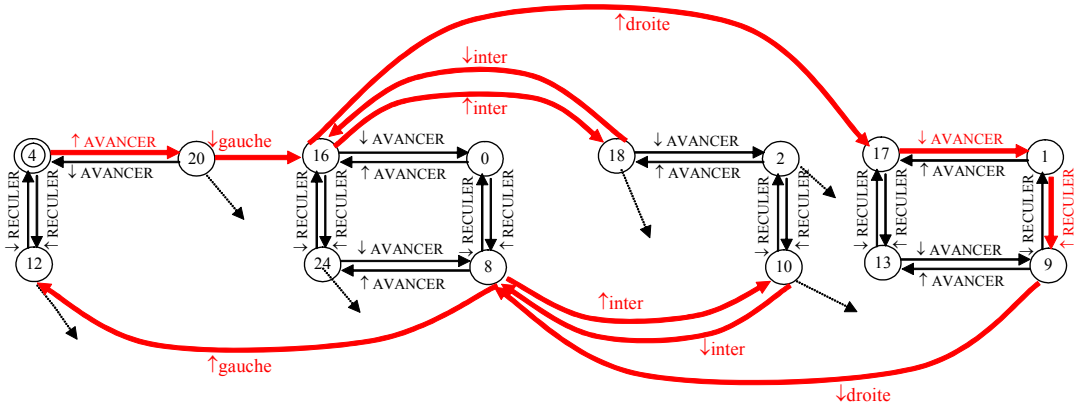


Figure 3.12 : Extrait de l'automate du mouvement du chariot.

3 MODELISATION DES CONTRAINTES

3.1 Contexte

Dans notre approche de synthèse, les contraintes à modéliser sont de deux types : des contraintes de sécurité, correspondantes à ce qu'il ne faut pas faire et des contraintes de vivacité correspondantes à 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. Dans les SED, les contraintes sont souvent exprimées par des automates à états. Ces automates, introduits pour la spécification des contraintes, peuvent être peu compréhensibles car la signification des états n'est pas toujours très explicite pour exprimer une situation dans la contrainte, et spécifiquement quand le nombre d'états de la spécification est très grand.

Par conséquent, pour faciliter l'écriture de ces contraintes sans utiliser des automates à états, nous préconisons une alternative basée sur l'utilisation d'équations logiques qui sont fonctions de l'ensemble des entrées et de l'ensemble des sorties du Grafcet. L'utilisation, pour l'expression des contraintes, des équations logiques en lieu et place d'automates à états, permet de réduire l'explosion combinatoire lors de la génération de la commande. Cependant, il faut préciser que l'expression d'une contrainte par équation logique demande un certain recul et une bonne connaissance du cadre formel de modélisation et de la complexité de la contrainte que l'on veut exprimer.

Il existe des travaux utilisant les équations logiques dans l'algèbre de Boole. Nous pouvons citer parmi ces travaux ceux de Roussel et *al* qui utilisent une approche algébrique basée sur les signaux binaires pour développer une méthode de synthèse formelle. Cette approche permet d'obtenir, pour un système logique, une commande conforme aux spécifications [ROU 03]. Elle repose sur la formalisation des spécifications sous forme d'assertions dans une algèbre de Boole adaptée aux signaux binaires appelé Algèbre *II* [ROU 94]. L'algèbre *II* permet de décrire des comportements intégrant des aspects temporels, en manipulant cette fois des signaux binaire fonction du temps et des variables booléennes. Par la suite, nous faisons dans notre démarche abstraction de l'aspect temporel et nous nous intéresserons seulement à l'utilisation de l'algèbre de Boole classique pour modéliser les contraintes.

3.2 Cadre formel de modélisation

Le cadre formel de la modélisation des contraintes est basé sur l'algèbre de Boole des propositions pour laquelle les éléments manipulés sont des valeurs binaires 0 ou 1 qui représentent les valeurs des entrées E et des sorties Z du Grafcet de spécification. Ces éléments sont représentés par des fonctions définies dans *IB*.

Pour composer les éléments de *IB* entre eux, trois lois de base sont définies :

Loi ET	*Loi OU*	*Loi NON*
$IB^2 \rightarrow IB$	$IB^2 \rightarrow IB$	$IB \rightarrow IB$
$(f, g) \rightarrow f \wedge g$	$(f, g) \rightarrow f \vee g$	$f \rightarrow \neg f$

En plus des connecteurs logiques ($\wedge$, $\vee$, $\neg$), il existe de nombreux connecteurs, parmi eux le connecteur implication ($\rightarrow$: SI … ALORS…). Ce connecteur peut être utilisé pour exprimer une vivacité.

Nous rappelons que, dans cette algèbre de Boole (*IB,* $\wedge$*,* $\vee$*,* $\neg$ *),* les propriétés suivantes (dites axiomes [PERM 88]) sont satisfaites pour tout f, g, h $\in$ *IB*

$f \wedge g = g \wedge f$	$f \vee g = g \vee f$	Commutativité
$f \wedge (g \vee h) = (g \wedge f) \vee (h \wedge f)$	$f \vee (g \wedge h) = (f \vee g) \wedge (f \vee h)$	Distributivité
$(f \wedge 1) = f$	$f \vee 0 = f$	Identité
$f \wedge \neg f = 0$	$f \vee \neg f = 1$	Inversion

Pour effectuer des calculs dans cette algèbre, un ensemble de théorèmes déduits sont utilisés.

$f \wedge f = f$	$f \vee f = f$	Idempotence
$f \wedge 0 = 0$	$f \vee 1 = 1$	Dominance
$f \wedge (f \vee g) = f$	$f \vee (f \vee g) = f$	Absorption
$f \wedge (g \wedge h) = (f \wedge g) \wedge h$	$f \vee (g \vee h) = (f \vee g) \vee h$	-Associativité

Les trois opérations de base *(ET, OU, NON)* combinent les valeurs booléennes

seulement dans la logique combinatoire, c'est-à-dire que la valeur d'une variable à un instant donné est obtenue par les valeurs des opérandes au même instant.

3.3 Equations logiques en termes d'entrées/sorties

On considère la machine séquentielle élémentaire, présentée dans la figure 3.13, qui doit produire des sorties $E = \{e_1, e_2, \ldots, e_i, \ldots, e_n\}$, en fonction des entrées $Z = \{z_1, z_2, \ldots, z_i, \ldots, z_n\}$

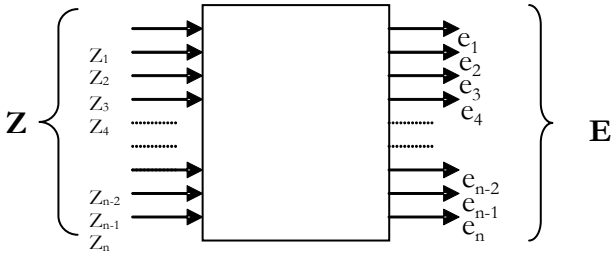


Figure 3.13. **Entrées/sorties du systèmes**

Pour passer formellement des spécifications entre les entrées et les sorties du système de commande à un ensemble d'équations exprimant les contraintes de sûreté de fonctionnement à respecter par le système, nous formalisons les contraintes sous forme algébrique par des relations liant les éléments des deux ensembles E et Z.

Dans l'algèbre de Boole, la modélisation des contraintes est classée en trois types : soit la contrainte peut être décrite par une combinaison logique *f* des entrées de la commande soit par une combinaison logique *g* des sorties de la commande où soit par la combinaison logique des entrées/sorties *Spec(f,g)*:

$$f : E \rightarrow IB \qquad g : Z \rightarrow IB \qquad SPEC : IB^2 \rightarrow IB$$

$$e \rightarrow f(e) \qquad z \rightarrow g(z) \qquad (f,g) \rightarrow Spec\ (f,g)$$

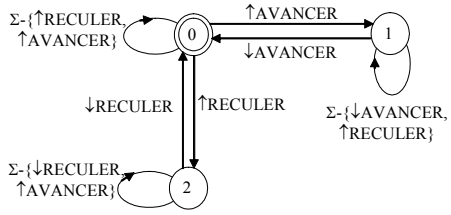
$$e \in E \qquad z \in Z$$

NB : On définit les mêmes lois de bases ($\vee$, $\wedge$, $\neg$) pour les trois fonctions *f,g* et *Spec*.

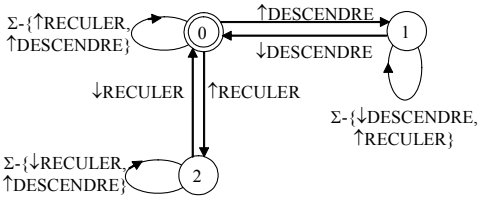
3.4 Illustration

Pour l'exemple du préhenseur, nous imposons quatre contraintes de sécurité et deux contraintes de vivacité liées à l'aspiration (chapitre 2). Les trois premières contraintes

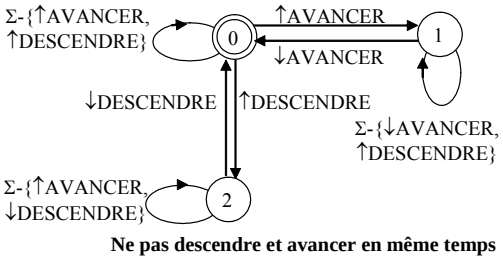
consistent à interdire l'activation simultanée des ordres *AVANCER* et *RECULER*, des ordres *DESCENDRE* et *RECULER*, et des ordres *AVANCER* et *DESCENDRE*. Les automates à états ainsi que leurs équations logiques équivalentes représentant ces contraintes sont donnés Figure 3.14.



❶*Si* AVANCER *Alors* ¬RECULER
où
AVANCER ∧ RECULER = 0



❷*Si* RECULER *Alors* ¬DESCENDRE
où
RECULER ∧ DESCENDRE = 0

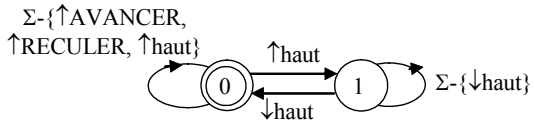


❸ *Si* AVANCER *Alors* ¬DESCENDRE
où

RECULER ∧ DESCENDRE = 0

Figure 3.14. Interdiction simultanée des ordres AVANCER ; RECULER ; DESCENDRE.

Pour des raisons liées à la sécurité du personnel et à la saisie du pignon sur son poste, la quatrième contrainte de sécurité étudiée n'autorise les mouvements horizontaux qu'en position haute et s'exprime par l'équation logique suivante (Figure 3.15).

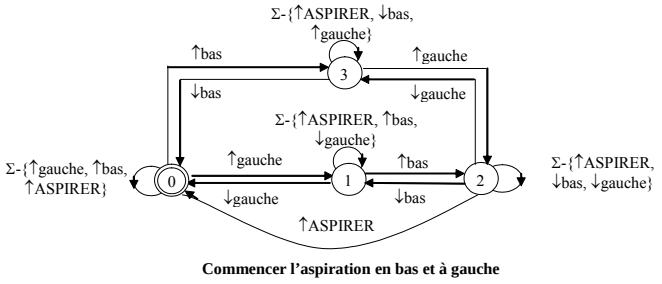


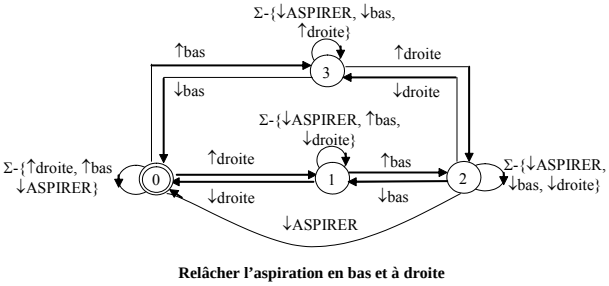
❹ *Si* AVANCER ou RECULER *Alors* haut
où
(AVANCER ∨ RECULER) ∧ ¬haut = 0

Figure 3.15. Ne pas avancer ou reculer sans être en position haute.

Les équations ❺ et ❻ décrivent les contraintes qui représentent la chronologie des événements autour de l'aspiration (figure 3.16) :

- La contrainte ❺ vérifie que l'ordre ASPIRER est envoyé lorsque le préhenseur se situe en bas à gauche.
- La contrainte ❻ vérifie que l'ordre ASPIRER est inhibé quand le préhenseur est en bas à droite.





❺ *Si* bas et gauche *Alors* ASPIRER
où
¬ASPIRER ∧ (bas ∧ gauche) =0

❻ *Si* droite et bas *Alors* ¬ASPIRER
ou
ASPIRER∧ (droite ∧ bas)= 0

Figure 3.16 Contraintes sur l'aspiration.

Le modèle global des contraintes obtenu par composition synchrone se compose de 128 états et 1856 transitions dans le cas d'une modélisation par automate à états. Dans le cadre de la modélisation des contraintes par des équations logiques, il suffit de spécifier cinq équations. Par conséquent, lors de la synthèse, on peut envisager une réduction très significative de l'espace mémoire pour l'élaboration de la commande.

4 CONCLUSION

Ce chapitre aborde la phase de modélisation de la partie opérative et des contraintes de notre démarche de synthèse de commande. Tout d'abord, nous nous sommes intéressés à la modélisation de la partie opérative vu son importance dans notre démarche et sa complexité. Dans la littérature, nous avons souvent constaté que la modélisation du comportement de la PO se faisait de façon intuitive et par automate à états, ce qui peut nuire à l'expressivité et la convivialité. Pour aider alors le concepteur dans la conception de ces modèles, nous avons proposé une méthodologie de modélisation structurée à base de règles. Cette dernière a permis alors de rendre la modélisation automatique et plus structurée. Deux méthodes sont développées, une méthode à base de construction par produit croisé synchrone entre l'automate de précédence et d'occurrence et l'autre méthode à base de construction booléenne. A l'issue de cette dernière méthode, le vecteur d'entrée E et de sortie Z associés à chaque état permettent d'enrichir le modèle de partie opérative en donnant une information plus précise sur l'état des capteurs et des actionneurs du système modélisé.

La méthode de modélisation de la partie opérative proposée n'est pas seulement applicable à des vérins pneumatiques, simple effet et double effet, comme illustré dans ce chapitre mais aussi à d'autres types d'actionneurs, comme par exemple, les moteurs électriques (plateau tournant à double sens par exemple (voir Annexe 3)), etc....

En deuxième partie de ce chapitre, nous avons proposé une modélisation des contraintes par équations logiques. Ce choix est justifié par la simplicité de l'expression et surtout par le fait que le concepteur maîtrise mieux la modélisation par équations logiques que par des automates à états. Dans le prochain chapitre, nous allons montrer comment la sémantique de chaque modèle est prise en compte pour synthétiser la commande correcte recherchée dans un but d'implantation.

Chapitre 4

SYNTHESE DE COMMANDE

Nous avons présenté dans le chapitre précédent une méthode d'aide à la conception des modèles de partie opérative (PO) par une modélisation structurée à base de règles. Pour les contraintes, nous avons utilisé une modélisation par équations logiques plus flexible que les automates à états, entre les entrées/sorties de la commande. Nous proposons donc d'adapter la démarche de synthèse pour prendre en compte cette nouvelle modélisation. Pour pallier à une erreur éventuelle de modélisation de la partie opérative, des contraintes, et du Grafcet de spécification mais aussi pour apporter une aide à l'élaboration de la commande, nous proposons d'intégrer le concepteur dans la boucle de synthèse. Ces différents apports sont ainsi traités dans ce chapitre.

1. INTRODUCTION

Dans le chapitre précédent, nous avons proposé des outils de modélisation plus évolués pour la partie opérative et les contraintes. Pour tenir compte de la différence entre la sémantique des modèles utilisés, équations logiques pour les contraintes, le grafcet pour la spécification de la commande et les automates à états booléens pour la partie opérative, il est nécessaire de développer une procédure d'intersection ❶ permettant d'extraire un modèle de commande qui respecte au mieux la réactivité et le déterminisme de la commande (figure 4.1).

Dans un premier temps, nous cherchons à mettre en œuvre la démarche intuitive et progressive définie dans la section 2 du chapitre 2 permettant de passer de la spécification à l'implantation de la commande. Cette mise en œuvre va montrer l'intérêt d'une modélisation par équations logiques et l'amélioration apportée à la démarche. Cependant, cette méthode a ses limites que nous allons expliquer dans la discussion de la section 2 de ce chapitre.

Dans un deuxième temps, nous allons reprendre la démarche formelle présentée dans la section 3 du chapitre 2 en étendant le processus de synthèse selon RW. L'utilisation de la théorie de RW impose une modélisation de la partie opérative et des contraintes par des automates rudimentaires. Nous utilisons désormais dans notre proposition une modélisation plus évoluée, par conséquent, nous proposons dans la section 3 une extension de la méthode de synthèse selon RW adaptée à la nouvelle modélisation.

Précédemment, nous avons vu que le traitement des blocages et les corrections effectuées avant l'implantation s'effectuent de façon automatique. Il consiste à rendre les états bloquants non atteignables par suppression de leurs transitions amont en cas de blocage et inhiber des actions en cas de corrections par rapport à la commande. Ces corrections automatiques sont transparentes pour le concepteur. En outre, pour lui procurer une véritable aide dans l'élaboration de sa commande et dans le raffinement des modèles de départ, nous lui proposons d'intervenir dans la démarche en ajoutant une étape de traitement semi-automatique. Cette étape se compose de deux aspects (figure 4.1), le premier correspond à la visualisation et à l'analyse des séquences bloquantes à travers une présentation de la trace des évolutions conduisant au blocage. Il

s'agit en fait d'interrompre l'étape de réduction au moment de la détection des blocages. De cette manière, le concepteur peut vérifier leur pertinence ou éventuellement agir sur les raisons du blocage en modifiant par exemple la commande, en relâchant des contraintes ou en affinant le modèle de la partie opérative. De nouvelles itérations de la synthèse sont alors effectuées à partir des modèles corrigés pour générer en finalité un modèle de commande correct [TAJ 03]. Le deuxième aspect consiste à appréhender une liste des contraintes non respectées et en déduire une explication par le concepteur. En effet, il se peut que les contraintes spécifiées entraînent des restrictions du comportement de la commande trop importantes ou non envisagées par le concepteur au moment de l'élaboration de son cahier des charges. La démarche lui permet alors de revenir sur son grafcet de commande et même d'ailleurs sur les spécifications des contraintes.

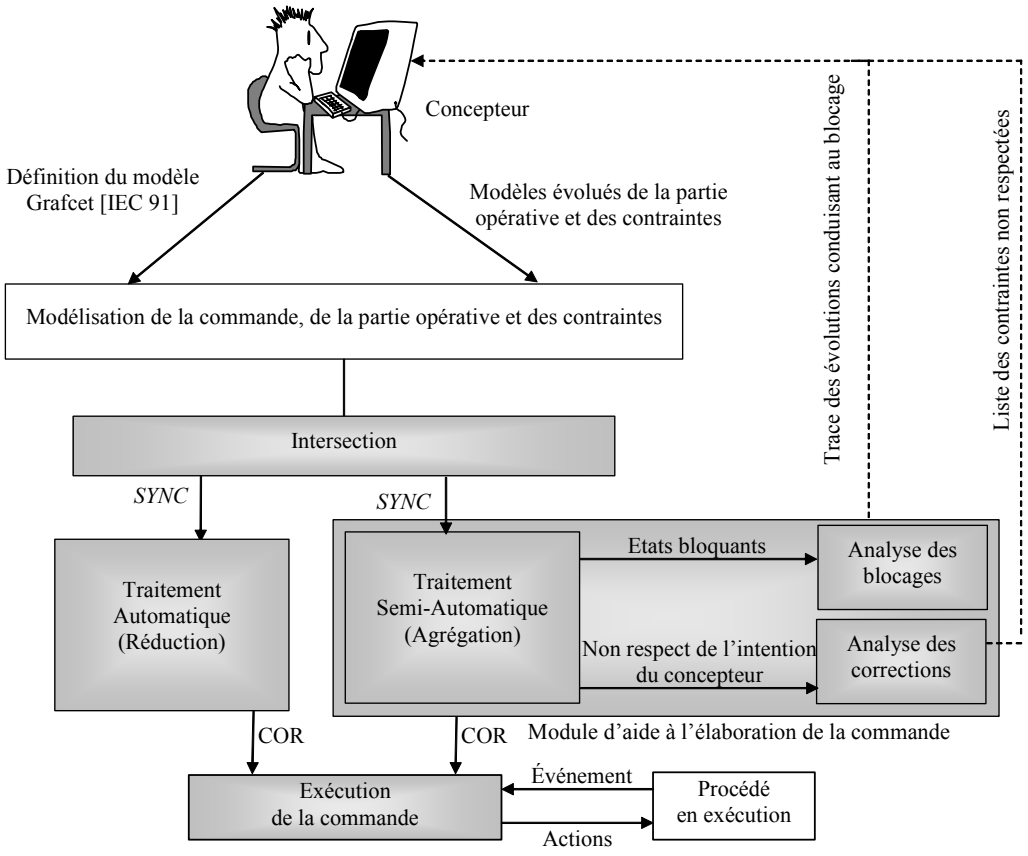


Figure 4.1. Démarche de synthèse de commande.

2. SYNTHESE DE COMMANDE : INTEGRATION DES CONTRAINTES DANS LA COMMANDE

Dans ce paragraphe, nous allons procéder à la synthèse de la commande selon le schéma de la figure 4.2. Il s'agit donc de développer des opérations d'intersection successives qui prennent en considération les différences sémantiques des différents modèles [TAJ 04]. Les détails des quatre étapes ont été présentés dans le chapitre 2.

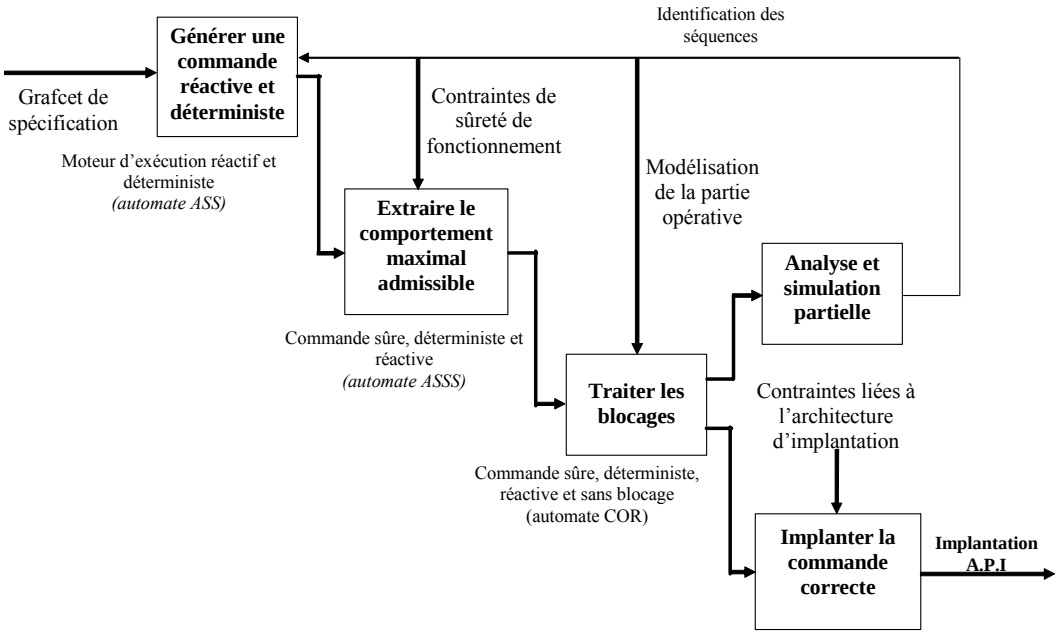


Figure 4.2. Opération de synthèse de commande Sûre

2.1. Intégration des contraintes dans l'automate des situations stables

De manière à assurer le bon fonctionnement de la commande décrite par le modèle ASS, il faut extraire le comportement le plus permissif de ce modèle vis-à-vis des contraintes. Il est donc nécessaire de développer une méthode permettant d'extraire un modèle de commande réactif et déterministe, respectant les contraintes exprimées par équations logiques.

Le principe de cette intégration, qui engendre l'Automate des Situations Stables et Sûres (ASSS), est particulier car le sens d'utilisation des contraintes et de ASS ne sont pas identiques. En effet, l'ASS, comme expliqué dans le chapitre 2, est défini par un automate caractérisé par des états et des transitions où à chaque état est associé un ensemble d'actions à envoyer à la PO et à chaque transition est associée une expression

logique qui permet de passer d'un état à un autre sur occurrence d'un évènement. Par contre, les contraintes sont caractérisées par des équations algébriques dépendant des actions et des entrées de la commande. Pour intégrer ces contraintes, nous avons identifié alors deux étapes qui sont :

- la formalisation sous forme algébrique des relations liant les éléments des deux ensembles E et Z (chapitre 3).

- L'analyse de ces différentes équations pour déterminer la dépendance de chaque contrainte (équation logique) par rapport à chaque ordre envoyé de la commande.

Soit K l'ensemble de toutes les contraintes et soit $Spec \in K$ une propriété à respecter lors de l'envoi d'un ordre $z \in Z$. Nous définissons un sous ensemble $K_z \subset K$ qui représente l'ensemble des équations à vérifier pour chaque ordre z tel que :

$$\forall z \in Z \text{ et si } \exists Spec\ (f(e),g(z)) \in K \text{ avec } e \in E \text{ alors : } Spec \in K_z.$$

Par exemple, pour le mouvement horizontal du préhenseur (chapitre 2), nous avons spécifié la contrainte « ne pas AVANCER et RECULER en même temps ». Cette contrainte est associée aux deux ordres *AVANCER* et *RECULER* et on écrit :

Ordre z	***Propriétés à respecter K_z***
AVANCER	$K_{AVANCER} = \{ \neg(\text{AVANCER} \wedge \text{RECULER})\}$
RECULER	$K_{RECULER} = \{\neg(\text{AVANCER} \wedge \text{RECULER})\}$

L'automate *ASSS* [TAJ 04], que nous obtenons par intégration des contraintes, est défini par un modèle sémantique similaire à *ASS* où les transitions correspondent à des variations d'entrées *f(E)*. Ces variations se définissent par des expressions logiques en fonction des entrées du Grafcet. Les ordres qui doivent être émis vers la partie opérative dans une situation stable du Grafcet, sont représentés dans *ASSS* par un ensemble Z_y constitué des ordres autorisés par les contraintes (figure 4.3).

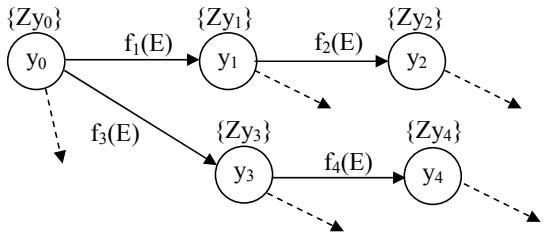


Figure 4.3. Modèle sémantique de l'automate *ASSS*

Formellement nous décrivons l'Automate des Situations Stables et Sûres par un 5-uplet *ASSS* = (E, Z, Y, T_y, y_0).

- E l'ensemble des entrées du Grafcet,

- Z l'ensemble des sorties du Grafcet,

- Y l'ensemble des situations stables et sûres du Grafcet. A chaque état y de Y on associe les deux ensembles suivants :

  - $Z_y \subseteq Z$ qui représente toutes les sorties actives dans cette situation.

  - $Etapes_y$ qui représente toutes les étapes du Grafcet actives dans cette situation.

- T_y : Y x f(E) $\rightarrow$ Y est une fonction partielle représentant les transitions de *ASSS* qui correspondent aux évolutions du Grafcet à l'échelle du temps externe. Une transition est représentée par un triplet (y, f(E), y') $\in$ T_y ; y étant l'état de départ, y' l'état d'arrivée et f(E) l'expression logique associée. Cette expression est composée des entrées du Grafcet et des fronts sur ces entrées [ROU 94].

- y_0 est l'état initial.

L'algorithme d'intégration des contraintes dans la commande reçoit en entrée un automate ASS = (E, Z, X, T, x_0), l'ensemble des contraintes *K* et la situation initiale (E, Z, {y_0}, $\varnothing$, y_0).

L'algorithme itératif d'intersection entre ASS et les contraintes présentées dans la figure 4.4, engendre un automate *ASSS* déterministe et réactif tout comme *ASS*.

$\forall y \in Y$ Faire :
  $\forall x \in X$ Faire :
  y=x ;
  $Z_y = \varnothing$;
  $Etape_y = Etape_x$;
  $T_y = T_x$;
    Si $Z_x \neq \varnothing$ **Alors**
        $\forall z \in Z_x$ Faire :
        $\forall$ *Spec* $\in K_z$ Alors :
        Si K_z est vraie **Alors**
          - $Z_y = Z_y \cup \{z\}$
          - $Z_x = Z_x - \{z\}$
        Sinon
          - $Z_x = Z_x - \{z\}$

Figure 4.4. Algorithme de génération de ASSS.

Chaque itération consiste à tester l'ensemble des propriétés K_z dans chaque situation courante *x* de l'automate *ASS* par rapport à une action *z*. Lorsque les différentes propriétés K_z associées à l'ordre *z* sont respectées, cet ordre est ajouté à Z_y.

Cet algorithme sert à créer les états des différentes situations stables du grafcet et les transitions qui les relient. L'ensemble des ordres envoyés vers le procédé représente les actions autorisées par les contraintes exprimées par des équations logiques. Autrement dit, cet algorithme consiste à créer, dans chaque état, l'ensemble des actions correspondant à une situation du grafcet et qui ne sont pas en contradiction avec les propriétés correspondantes à chaque action envoyée. Les transitions entre les situations décrivant les variations d'entrées qui lient une situation stable à une autre, ne subissent pas de changement.

2.2. Prise en compte de la partie opérative

La prise en compte de la partie opérative pour générer au final une commande sûre, déterministe, réactive et sans blocage sous la forme d'un automate appelé *COR* nécessite deux étapes : une étape dite d'intersection et une étape de traitement de blocages.

2.2.1. Intersection

L'automate *ASSS* calculé sans considérer les contraintes introduites par la partie opérative décrit un sur-ensemble du comportement effectif de la commande au sein du système global. En effet, la partie opérative introduit de nombreuses contraintes sur

l'évolution des entrées et restreint, par conséquent, le comportement de la commande lors de l'exécution. Ces contraintes peuvent être :

i) statiques, engendrées par la structure de la partie opérative. Par exemple, les entrées correspondantes aux fins de course *haut* et *bas* du préhenseur ne peuvent être vraies en même temps.

ii) dynamiques, caractérisant les scénarios d'évolution des entrées de la commande qui dépendent de la dynamique de la partie opérative et de ses interactions avec le Grafcet.

Les évolutions de *ASSS* ne répondant pas à ces scénarios doivent être éliminées pour tendre vers le comportement effectif de la commande Grafcet.

L'intersection d'un modèle de partie opérative avec l'automate ASSS permet d'aboutir à un automate global reflétant l'exécution réelle du système si la partie opérative est correctement modélisée. Les blocages identifiés sur cet automate correspondent donc à des blocages réels, susceptibles d'avoir lieu lors de l'exécution de la commande. L'idée de l'intersection est de parcourir l'automate de partie opérative et l'automate *ASSS* pour construire ensuite l'automate d'intersection à partir des événements admissibles par les deux automates.

L'élaboration de l'intersection est assez particulière car chaque automate repose sur une sémantique propre. Les états de ASSS reflètent les ordres à émettre et les transitions sont décrites par une fonction composée des entrées du Grafcet et de fronts sur ces entrées. Quant au modèle de PO, il correspond à un automate rudimentaire, où les transitions sont décrites par des événements.

Principe :

L'automate SYNC se construit alors en tant que modèle asynchrone dont les transitions correspondent à des événements simples. Les ordres qui doivent être émis en parallèle dans une situation du Grafcet, sont représentés dans SYNC par une structure arborescente composée de transitions décrivant les activations et les désactivations successives des ordres envoyés de l'automate ASSS. Les états reliés par ces transitions constituent une région qui corresponde à la dite situation du grafcet. Ainsi, l'automate SYNC sera constitué d'un ensemble de régions. Chaque région correspond à une situation du Grafcet ce qui permet d'exprimer le parallélisme de la commande. Par

exemple, pour une situation du grafcet correspondant à l'envoi de deux ordres simultanés, l'automate SYNC va engendrer une région à quatre états correspondant aux ordres parallèles (figure 4.5). Les transitions intra-régions correspondent aux événements commandables, et les transitions inter-régions aux événements non commandables (figure 4.6).

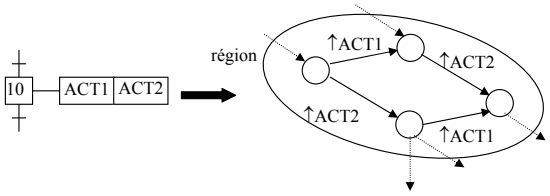


Figure 4.5. Exemple de conservation du parallélisme dans l'automate SYNC

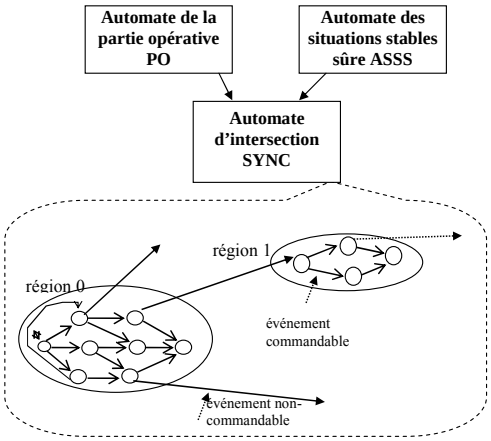


Figure 4.6. Intersection entre l'automate ASSS et PO.

Formellement, l'automate SYNC est définit par un 6-uplet ($\sum$, ST, TR, st_0, r_0, BLOC) avec :

- $\sum = \sum_c \cup \sum_u$,
- ST est l'ensemble des états de SYNC partitionnés en sous ensembles appelés régions. Un état $st \in ST$ correspond à une configuration unique du 3-uplet (q, y, E), où q est un état de PO, y un état ASSS et E l'ensemble des valeurs logiques (0 ou 1) des entrées du Grafcet. Tous les états correspondant à une situation du Grafcet et à une configuration unique de ses variables d'entrée, sont groupés dans une région. Chaque région $r \subset ST$ est munie de l'ensemble $Etapes_r$ qui contient les étapes actives dans la situation correspondante du Grafcet.
- $TR : ST \times \sum \rightarrow ST$ est une fonction partielle représentant les transitions de SYNC. Une transition est définie par un triplet $(st, \sigma, st') \in TR$.
- $st_0 \in ST$ est l'état initial qui est donné par (q_0, y_0, E_0), où q_0 est l'état initial de PO, y_0 l'état initial de ASSS et E_0 correspondants à la situation initiale y_0 du Grafcet et aux valeurs initiales E_0 de ses variables d'entrée.

- r_0, la région initiale, qui inclut l'état initial st_0. Elle est munie de l'ensemble des étapes correspondantes à la situations initiales du Grafcet Etapes $_{r0}$ = Etapes $_{y0}$.
- BLOC $\subset$ ST contient l'ensemble des états bloquants, c'est-à-dire ceux n'ayant pas de transitions sortantes. Dans chacun de ces états, les actions et/ou les évolutions de la commande sont en contradiction avec les événements issus des états correspondants de la partie opérative.

L'automate SYNC est généré par un algorithme itératif basé sur deux étapes, servant respectivement au développement de la région initiale r_0, et au développement d'une région quelconque r_c et des transitions inter régions. Cet algorithme s'arrête lorsque aucune nouvelle région ne peut plus être créée. Il reçoit en entrée un automate PO = (Σ, Q, δ, q_0) correspondant au comportement de la partie opérative, un automate ASSS =(E, Z, Y, T, y_0) équivalent du grafcet et la structure initiale de l'automate SYNC donnée par (Σ, {st0}, $\varnothing$, st_0, {st_0},$\varnothing$). Les détails de construction de l'automate SYNC sont présentés dans l'Annexe 1.

2.2.2. Traitement des blocages

L'objectif est maintenant de générer la commande réactive et déterministe (automate *COR*) correspondante au plus grand sous-ensemble de comportements de *SYNC*, exempt de blocage. Cette opération s'effectue en deux phases : une phase de retrait des états bloquants et une phase d'optimisation.

Dans la première phase, les états bloquants de l'automate d'intersection sont rendus non atteignables par suppression de leurs transitions amont. Ces suppressions pouvant induire de nouveaux états bloquants, ceux-ci sont alors traités de manière récursive. La deuxième phase consiste à retirer de l'automate d'intersection, issu de la première phase, les transitions qui sont non atteignables durant l'exécution réelle, à cause de la nature réactive de la commande, et à agréger les situations instables (Annexe 1), de manière à aboutir à une commande réactive qui évolue, suite à l'occurrence d'un événement non commandable, d'une situation stable à une autre. Par conséquent, une situation stable correspond à une agrégation des états appartenant à une région de l'automate d'intersection suite au traitement du blocage ; elle contient les informations relatives aux ordres qui sont autorisés ou inhibés de manière instantanée, au sein de cette région.

A chaque état de *COR* est associé le quadruplet (Z_c, Sit_c, ORD, PROH) où Z_c correspond à l'ensemble des sorties du Grafcet lié à cet état et Sit_c l'ensemble des étapes Grafcet correspondant à cet état. *ORD* et *PROH* représentent l'ensemble des ordres, calculés dynamiquement, à envoyer (respectivement à interdire) par *COR*. Ces ensembles sont utilisés pour renseigner l'utilisateur sur les ordres maintenus ou non par la procédure de synthèse.

Les algorithmes de traitement de blocage, d'optimisation et d'agrégation sont présenté dans l'annexe 1.

2.3. Illustration

En utilisant l'exemple du préhenseur, nous allons illustrer le principe d'intégration des contraintes modélisées par des équations algébriques dans la commande spécifiée par grafcet (figure 4.7).

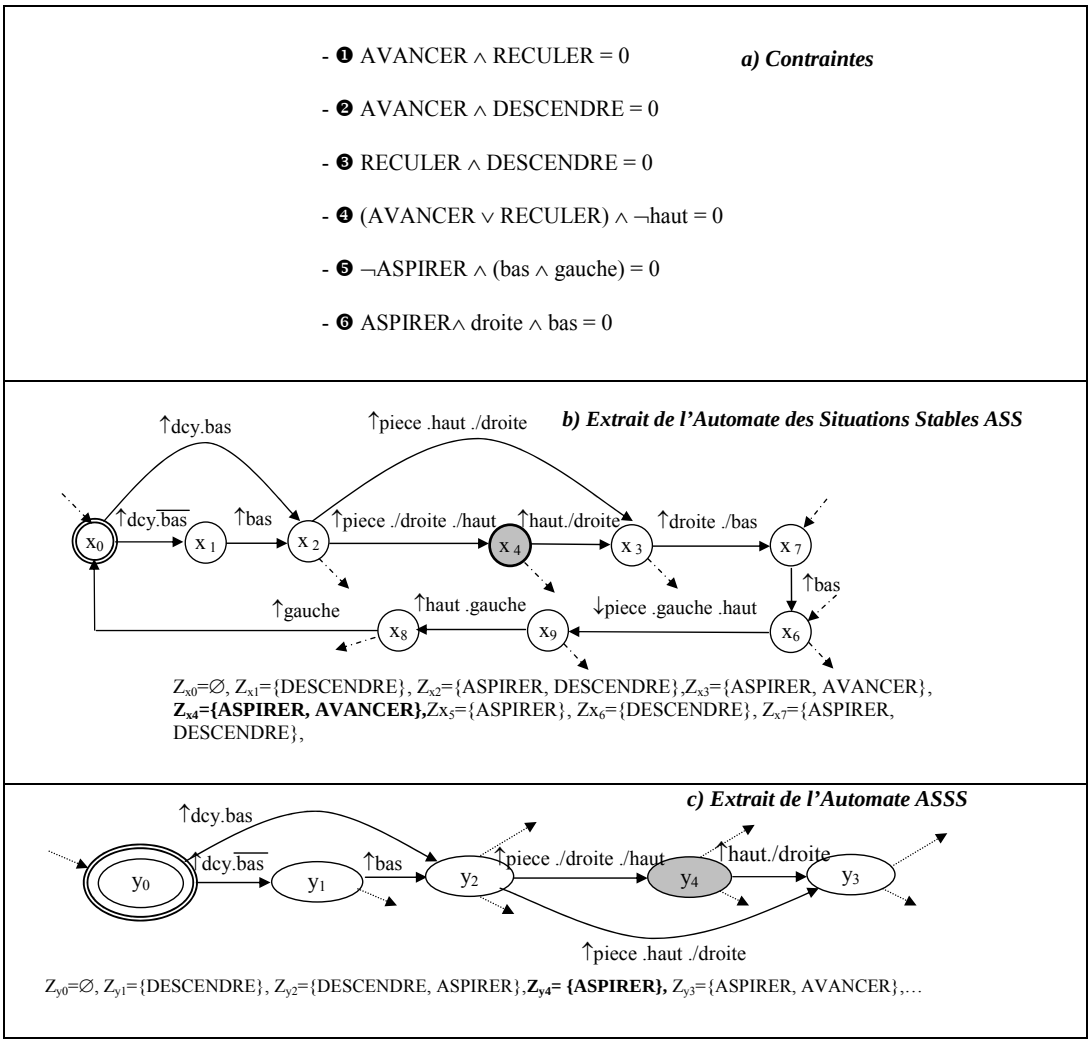


Figure 4.7. Illustration de la construction de l'Automate ASSS.

Le tableau 4.1 présente les différentes contraintes associées à chaque sortie qui doit être vérifiée pour chaque situation de la commande grafcet.

Sorties du Grafcet z	***Propriétés K_z à respecter***
AVANCER	$\boldsymbol{K_{AVANCER}}$ **= {❶, ❷, ❹}**
RECULER	$\boldsymbol{K_{RECULER}}$ = {❶, ❸, ❹}
DESCENDRE	$\boldsymbol{K_{DESCENDRE}}$ = {❷, ❸}
ASPIRER	$\boldsymbol{K_{ASPIRER}}$ = {❺, ❻}

Tableau 4.1. Dépendance des actions des contraintes spécifiées.

2.3.1. Intégration des contraintes dans la commande

L'état initial y_0 de l'automate *ASSS* (figure 4.7c) issu de l'intersection correspond à l'état initial de l'automate des situations stables *ASS* où aucune action n'est à envoyer par la commande ou à interdire par les contraintes.

- A partir de x_0, seule l'évolution caractérisée par la variation d'entrée *↑dcy./bas* permet d'atteindre la situation x_1. Cette situation x_1 est caractérisée par l'envoi de l'ordre *DESCENDRE* et puisque les contraintes ❷ et ❸ sont vérifiées dans cette situation alors l'ordre sera également disponible dans la situation y_1. L'état y_1 est donc défini par $Z_{y1} = Z_{y1} \cup \{DESCENDRE\} = \{DESCENDRE\}$.
- Deux évolutions distinctes sont alors possibles pour atteindre la situation x_2, soit par la variation d'entrée *↑dcy.bas* à partir de x_0, soit à travers la variation d'entrée *↑bas* à partir de x_1. Dans la situation x_2, les deux sorties *ASPIRER* et *DESCENDRE* sont actives. Pour *ASPIRER*, les contraintes ❺ et ❻ sont vérifiées et donc $Zy_2 = Zy_2 \cup \{ASPIRER\}$. Pour *DESCENDRE* les deux contraintes ❷ et ❸ sont aussi vérifiées car ni *AVANCER* ni *RECULER* ne sont envoyés. En conclusion, l'état y_2 autorise les deux ordres *DESCENDRE* et *ASPIRER* tel que :

  $$Zy_2 = \{ASPIRER, DESCENDRE\}.$$

- La variation d'entrée conduisant de la situation x_2 à la situation x_4 est liée à *↑pièce./droite./haut.* Dans cette situation, les deux ordres *ASPIRER* et *AVANCER*

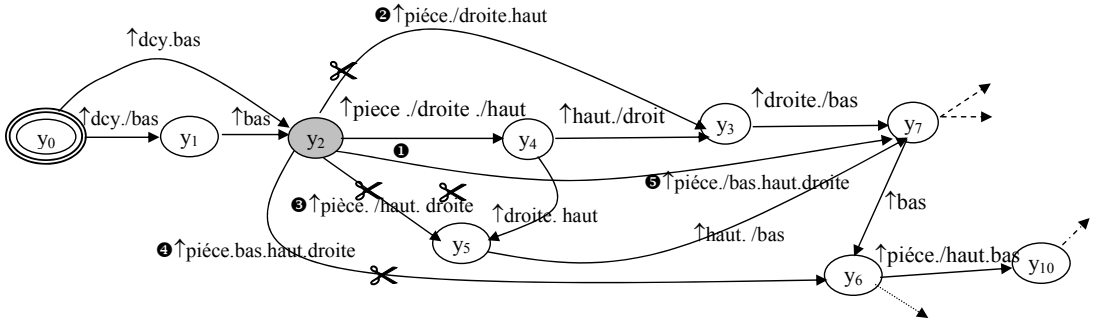
sont actifs. Pour *ASPIRER* les contraintes ❺ et ❻ sont vérifiées et donc $Z_{y4}= Z_{y4} \cup \{ASPIRER\}$. Par contre, dans cette situation, l'ordre *AVANCER* est interdit car la contrainte ❹ exige pour le préhenseur d'être en position haute pour autoriser l'ordre *AVANCER* ce qui n'est pas le cas dans la situation x_4. Par conséquent, l'état de l'automate *ASSS* donné par y_4 est caractérisé par :

$$Z_{x4}= Z_{x4}-\{AVANCER\} = \{ASPIRER\}$$

$$Z_{y4}= Z_{y4} \cup \{ASPIRER\}.$$

2.3.2. Prise en compte de la partie opérative

La situation y_2 évoquée précédemment, présente un exemple (figure 4.8) qui illustre l'impossibilité d'occurrence de nombreuses évolutions de *ASSS* pendant l'exécution réelle. Parmi les cinq transitions possibles à partir de cette situation seule la transition notée❶ correspond à un comportement possible de la commande en exécution réelle. Les quatre autres transitions ne peuvent se produire en réalité parce qu'elles correspondent à des évolutions irréalistes telles que la présence simultanée des deux variables *haut* et *bas* (transition❹), la présence de *droite* (transitions ❸ et ❺) ou de *haut* (transitions ❷ et ❺) alors que le préhenseur se situe en *bas* et à *gauche*.



$Z_{y0}=\varnothing$, Z_{y1}={DESCENDRE} Z_{y2}={ASPIRER, DESCENDRE},},Z_{y3}={ASPIRER, AVANCER}, **Z_{y4}={ASPIRER},…**
Zy_5={ASPIRER}, Zy_6={DESCENDRE}, Z_{y7}={ASPIRER, DESCENDRE},Z_{y8}={RECULER}, $Z_{y9}= Z_{y10}=\varnothing$.

Le vecteur d'entrée : E= [gauche, droite, haut, bas, dcy, pièce].

Le vecteur de sortie : Z= [AVANCER, RECULER, DESCENDRE, ASPIRER].

Figure 4.8. Extrait de l'Automate ASSS.

2.3.2.1. Intersection entre la PO et l'ASSS

a. Développement de la région initiale

La région r_0 (tableau 4.2) représente la région initiale de l'automate d'intersection, elle est développée à partir de l'état initial (q_0, y_0, E_0) où q_0 représente l'état initial de la

partie opérative, y_0 l'état initial de *ASSS* et E_0 = [1, 0, 1, 0, 0, 0] le vecteur d'entrée. Les sorties actives à l'instant initial sont données par l'ensemble $Z_{y0} = \varnothing$. Aucun événement commandable ne se produisant dans la situation y_0 de *ASSS*, la région initiale ne comporte donc qu'un seul état (tableau 4.2).

Partie opérative PO	**Evolution**		
	Etat amont	**Evénement commandable**	**Etat aval**
	q_0	↑AVANCER	q_3
		↑RECULER	q_4
		↑DESCENDRE	q_5
		↑ASPIRER	q_1

ASSS : automate des situations stables sûres	**Situation Grafcet**	**Actions du Grafcet**
	y_0	∅

Région initiale de l'Automate SYNC	**Intersection entre événement commandable et les actions**	**Région r_0**		
		Etat de la PO	**Etat de ASSS**	**Vecteur d'entrée**
	∅	**q_0**	**y_0**	**E_0 = [1.0.1.0.0.0]**

Tableau 4.2. Construction de la région initiale.

b. Développement des transitions inter-régions

Il faut créer à partir de *(q_0, y_0, E_0)*, les transitions de sortie inter-régions supportées par des événements non commandables. L'une des transitions sortantes de q_0 est l'événement non commandable *↑dcy* qui peut se produire dans l'état y_0 de *ASSS*. Cet événement valide l'expression logique de la transition reliant y_0 à y_1 dans *ASSS* et par conséquent, l'état initial de la nouvelle région r_1 est *(q_2, y_1, E_1)* avec E_1 = [1, 0, 1, 0, 1, 0] (tableau 4.3).

Partie opérative PO	**Evolution**		
	Etat amont	**Evénements non commandables**	**Etat aval**
	q_0	↑dcy	q_2

ASSS : automate des situations stables sûres	**Situation Grafcet**	**Transitions sortantes**	**Situations atteintes**
	y_0	dcy.bas	y_2
		dcy./bas	y_1

Région 3 de l' Automate SYNC	**Expressions logiques valides par ↑dcy**	**Etat atteint de la région r_1**		
	dcy./bas	**Etat de la PO**	**Etat de ASSS**	**Vecteur d'entrée**
		q_2	**y_1**	**E_1= [1.0.1.0.1.0]**

Tableau 4.3. Construction d'une transition inter région (*↑dcy*).

c. Développement d'une région quelconque

Comme *↑ASPIRER* représente l'unique événement commandable se produisant dans la situation courante, y_2, de *ASSS* et qu'il est autorisé dans la situation courante q_2 de la PO (figure 4.9.a et figure 4.9.b), la région r_3 inclut une transition correspondante à

l'occurrence d'*ASPIRER*, reliant l'état (q_{80}, y_2, E_3) au nouvel état *(q_{125}, y_2, E_3)* (tableau 4.4).

Partie opérative PO	**Evolution**		
	Etat amont	**Evénements commandables**	**Etat aval**
	q_{80}	↑AVANCER	q_{94}
		↑RECULER	q_{96}
		↑DESCENDRE	q_{97}
		↑ASPIRER	q_{125}

ASSS : automate des situations stables sûres	**Situation Grafcet**	**Actions du Grafcet**
	y_2	ASPIRER

Région 3 de l' Automate SYNC	**Intersection entre événement commandable et les actions**	**Région r_3**	
		Etat amont	**Etat aval**
	↑ASPIRER	**(q_{80}, y_2, E_3)**	**(q_{125}, y_2, E_3)**

Tableau 4.4. Construction d'une nouvelle région (r_3).

La figure 4.9 résume la démarche de construction des régions.

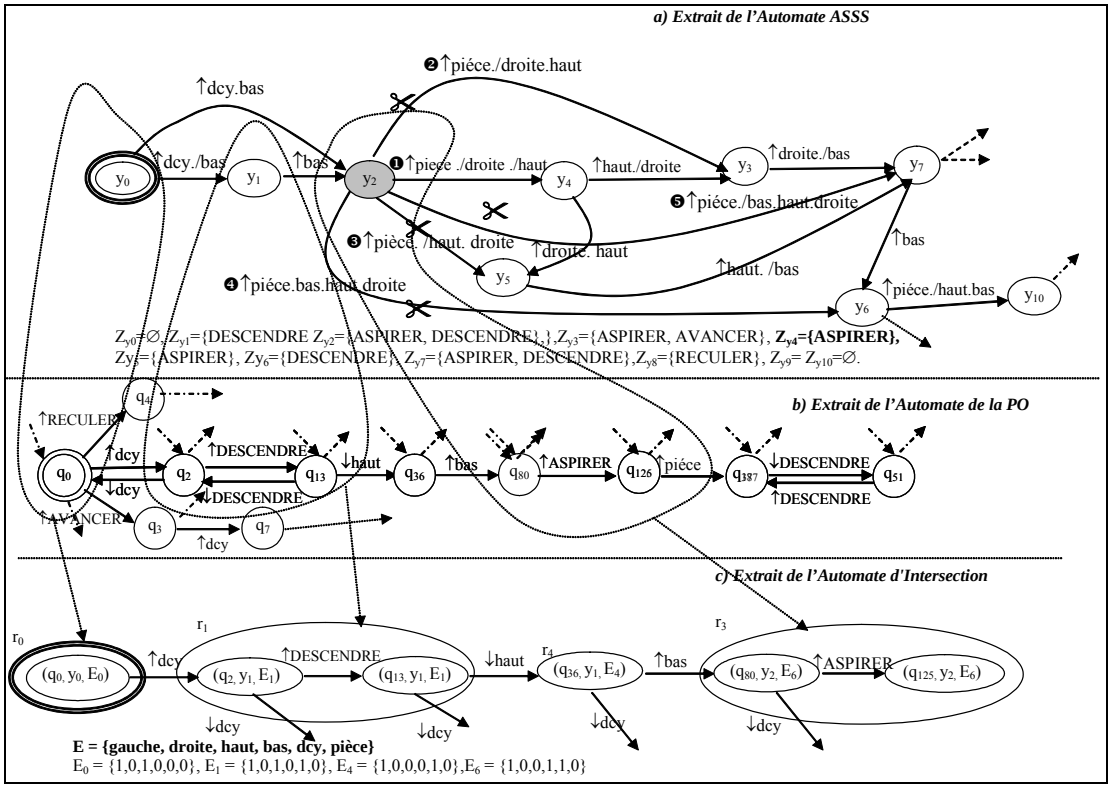


Figure 4.9. Les différents modèles liés aux étapes de synthèse de l'automate SYNC.

2.3.2.2. Obtention de l'automate COR

La procédure de réduction conduit à l'obtention de l'automate *COR* de la figure 4.10b. Pour cet exemple, aucun blocage n'a été détecté et les corrections paraissent

évidentes. En effet, dans l'état 8, l'ordre *AVANCER* est interdit ($PROH_8$ = {*AVANCER*}), alors qu'il est autorisé dans l'état 12. En observant de plus près les évolutions entre l'état 8 et l'état 12, on remarque que le système atteint la position haute, avant de recevoir l'ordre *AVANCER*. Cette inhibition d'ordre est liée à la prise en compte de la contrainte « *Ne pas avancer ou reculer sans être en position haute* ».

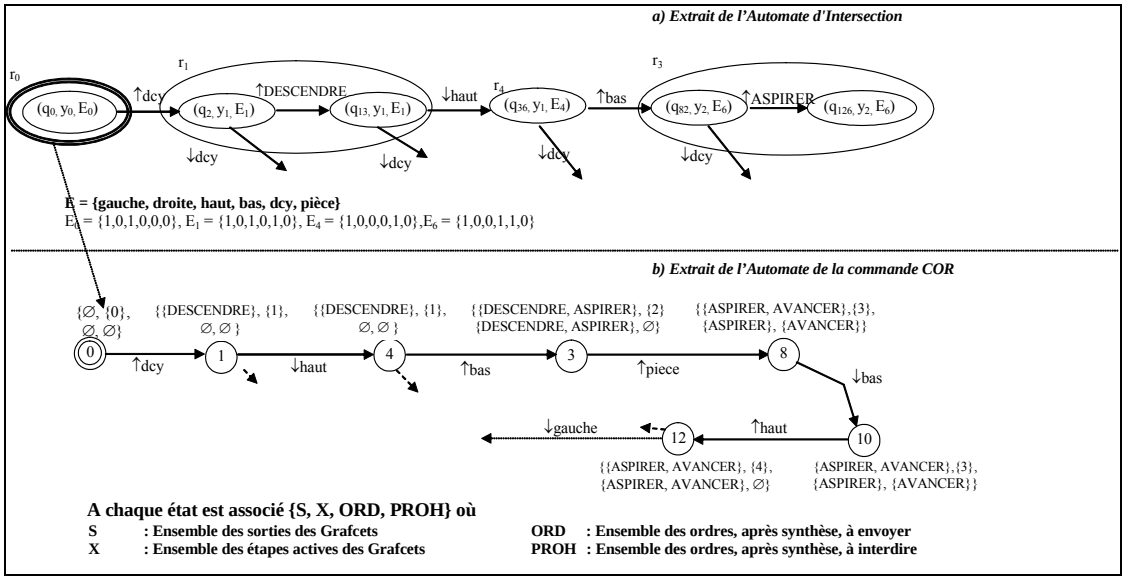


Figure 4.10. Génération de la commande COR.

2.4. Discussion

L'intégration des contraintes sous forme d'équations logiques à la place des automates à états, induit comme prévu, une réduction de l'explosion combinatoire lors de la démarche de synthèse.

Soit *l* le nombre d'état dans chaque automate de contrainte et *k* le nombre de contraintes spécifiées et m est le nombre d'états de l'automate *ASS*, le nombre d'états est dans le pire des cas proportionnel à $m.l^k$. Par contre, en intégrant les contraintes modélisées par des équations logiques dans la commande, le nombre d'état dans le pire des cas est proportionnel à *m.k,* ceci montre donc que le nombre d'états par cette modélisation est polynomiale par rapport à la modélisation événementielle qui est exponentielle.

Nous avons procédé pour réaliser une comparaison du calcul de *ASSS* à deux tests. Le premier test consiste à intégrer les contraintes modélisées par automates à états et le deuxième par équations logiques. Le tableau 4.5 illustre l'influence de la modélisation des contraintes par équations logiques sur la démarche de synthèse.

	Modélisation par des automates à états finis	Modélisation par des équations logiques
Automate ASSS	873 états	11 états
	7101 transitions	23 transitions

Tableau 4.5 : Influence des équations logiques sur l'opération de synthèse.

Malgré la réduction en terme d'explosion combinatoire, cette démarche a des limites qui peuvent nuire à sa praticabilité :

- L'utilisation de plusieurs opérations d'intersection rend la démarche lourde à concevoir.
- La vérification des contraintes est spécifique à celles liées aux ordres actifs dans chaque situation, par contre les contraintes liées aux ordres non activés ne sont pas vérifiées.

Pour résoudre ce problème, il faut donc prendre en compte l'ensemble des événements commandables $\sum_c = \uparrow Z \cup \downarrow Z$, c'est-à-dire les ordres en activation et en désactivation. En effet, l'ASS est défini par un ensemble d'actions en activation associé à chaque état sans aucune information sur les ordres non actifs et pour prendre en compte ces ordres, il faut donc ajouter un autre ensemble définissant les ordres non actifs et revoir l'algorithme d'intégration des contraintes, rendant alors la méthode plus complexe.

Nous avons montré précédemment que la théorie de supervision permet d'agir sur les événements commandables en interdisant ou en les autorisant dans le cas d'activation ou en désactivation. L'apport de cette théorie permet donc de pallier le problème présenté en prenant en compte toutes les contraintes de sûreté et de vivacité. Cependant, l'utilisation de cette théorie dans le nouveau contexte de modélisation nécessite de développer une extension de la méthode de synthèse selon RW du superviseur. Nous présentons plus explicitement cette extension dans la section suivante de ce chapitre.

3. SYNTHESE DE COMMANDE SUPERVISEE SOUS CONTRAINTES SPECIFIEE PAR DES EQUATIONS LOGIQUES

3.1. Contexte

L'utilisation des automates rudimentaires pour la synthèse du superviseur avec l'un des algorithmes comme celui de Kumar [KUM91] adopté dans notre démarche, génère une explosion combinatoire qui est, dans le pire des cas en supposant que tous les automates ont le même nombre d'états, proportionnelle à $N^n.M^k$. N présente le nombre d'états de chaque automate du procédé, n est le nombre d'automates, M est le nombre d'états dans chaque contrainte et k représente le nombre de contraintes. Nous proposons dans cette section d'adapter la démarche en 6 étapes (chapitre 2) en prenant en compte les nouveaux modèles (automate à états booléens pour la partie opérative et les équations logiques pour les contraintes) et en proposant une extension de l'algorithme de synthèse du superviseur (figure 4.11).

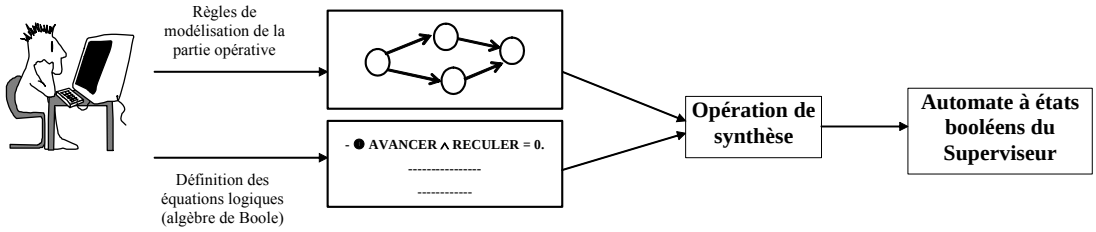


Figure 4.11. Etapes de la démarche de synthèse basée sur des modèles booléens.

Cette extension est construite autour d'une prise en compte spécifique des contraintes modélisées par des équations logiques. Ces contraintes sont structurées en fonction des événements commandables Σ_c représentant l'activation ou la désactivation des ordres $z \in Z$ envoyés de la commande vers la partie opérative. En effet, à chaque événement commandable $s \in \sum_c$ est associé un ensemble de contraintes K_s. Ces contraintes sont définies par des équations logiques dans l'algèbre de Boole. Soit K l'ensemble de toutes les contraintes et soit $Spec \in K$ une propriété à respecter suite à l'occurrence d'un événement commandable $s \in \sum_c$. Nous définissons un sous ensemble $K_s \subset K$ qui représente l'ensemble des équations à vérifier pour chaque s tel que :

$\forall s \in \sum_c$ et **Si** $\exists Spec \in K$ avec $e \in E$ tel que $Spec$ est vraie suite à l'occurrence de s

Alors : $Spec \in K_s$.

Par exemple, pour le mouvement horizontal de l'exemple du préhenseur, nous avons spécifié la contrainte « *Ne pas AVANCER et RECULER en même temps* ». Cette contrainte est associée à deux événements $\uparrow AVANCER$ et $\uparrow RECULER$ et s'écrit sous la forme suivante:

Evénements commandables 's'	***Propriétés à respecter K_s***
$\uparrow AVANCER$	$\boldsymbol{K}_{\uparrow AVANCER} = \{\neg(\text{AVANCER} \wedge \text{RECULER})\}$
$\uparrow RECULER$	$\boldsymbol{K}_{\uparrow RECULER} = \{\neg(\text{AVANCER} \wedge \text{RECULER})\}$

3.2. Synthèse du superviseur

3.2.1. Principe

Dans cette partie, nous cherchons à synthétiser un superviseur en utilisant les modèles automates à états booléens (chapitre 3 section 2.2.3) pour le procédé ($G = (P, \sum, \delta, p_0)$) et les équations logiques pour les contraintes *K*. Le principe consiste à interdire un événement contrôlable dans certains états pour empêcher le système de se rendre vers des états qui ne satisfont pas les spécifications. L'un des concepts les plus importants dans la théorie de supervision est celui de la contrôlabilité [RW 87](chapitre 1). Dans notre cadre, le principe de contrôlabilité est le suivant :

$$(\forall p \in P, \textit{tel que } p \textit{ vérifie } K)$$
$$\forall \sigma \in \sum_u \textit{ tel que } \delta(p, \sigma) \textit{ vérifie } K$$
$$\textit{Alors } K \textit{ est commandable}$$

La spécification *K* est commandable si et seulement si l'occurrence d'un événement non contrôlable σ 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*.

Nous avons vu que dans la section 1 du chapitre 1, lorsque le langage décrit par la spécification *K* est commandable et inclus dans le langage du procédé *L(G)*, il peut être utilisé comme superviseur. Dans le cas contraire, il n'existe pas de superviseur *SUP* tel que *L(S/G) = K*. Par conséquent, il faut chercher une solution plus restrictive, i.e. le plus grand sous-ensemble de *K* qui soit unique et commandable *supC(K)*, et qui respecte les restrictions imposées par *K*.

Dans le cas d'une modélisation par automates à états rudimentaires, l'un des algorithmes les plus utilisés pour cet objectif est celui de Kumar [KUM 91]. Nous proposons d'étendre cet algorithme de synthèse aux modèles retenus pour la PO et les contraintes.

3.2.2. Algorithme de synthèse

L'automate *SUP* est décrit par 6-uplet $(\Sigma, Q, \Delta, E, Z, q_0)$, 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'entrée E, et un vecteur de sortie Z. Une transition de l'automate *SUP* est définie par un triplet $(q, \sigma, q') \in \Delta$, q étant l'état de départ, σ l'événement correspondant à la transition et q' l'état d'arrivée. On note aussi qu'à l'état initial q_0 les deux vecteurs d'entrées et de sorties sont définis par E_0 et Z_0. La fonction de transition peut être étendue aux séquences d'événements de la manière suivante :

$$\Delta : Q \times \Sigma^* \rightarrow Q \text{ où } \Delta(q,\varepsilon) = q \text{ avec } \varepsilon \text{ est l'élément vide}$$

$$\Delta(q,w\sigma) = \Delta(\Delta(q, w), \sigma))$$

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

Cet algorithme (figure 4.12.a, b et c)) est basé sur trois étapes :

- La première étape de cet algorithme (figure 4.12.a), traite les événements commandables liés à l'activation ($\uparrow z$) ou à la désactivation ($\downarrow z$) décrivant les ordres envoyés à la partie opérative. Une itération consiste, pour une évolution commandable $((q, s, q'), (s \in \Sigma_c))$, à vérifier l'ensemble des contraintes K_s spécifiées par les équations logiques liées à l'événement s. Cette vérification est obtenue à travers la valeur du vecteur d'entrée E_q, du vecteur de sortie 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. Ceci signifie que cet ordre est prévu par la commande et autorisé par le superviseur *SUP*.

L'état aval *q',* atteint depuis q suite à l'occurrence de s*,* se distingue de l'état amont q par le vecteur de sortie $Z_{q'}$. L'état *q'* est ensuite ajouté à l'ensemble des états *Q* s'il n'en fait pas déjà partie. Par contre, si l'ensemble des contraintes K_s n'est pas vérifiée (les équations ne sont pas vraies), alors l'évolution (q, s, q') est défendue.

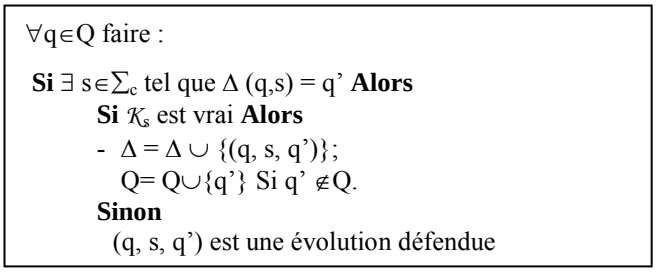


Figure 4.12.a. Traitement des événements contrôlables (1ère étape).

• La deuxième étape consiste à construire s'il n'existe pas déjà, à partir d'un état courant p, 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 interdits où les contraintes *K* ne sont pas vraies. Pour cet état q, il existe un événement non contrôlable σ possible à partir de l'état correspondant du procédé q' tel que l'état atteignable ne vérifie pas *K*. D'après la définition de la commandabilité, si le comportement du système ne contient aucun état défendu alors il est commandable et peut servir en tant que superviseur.

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.

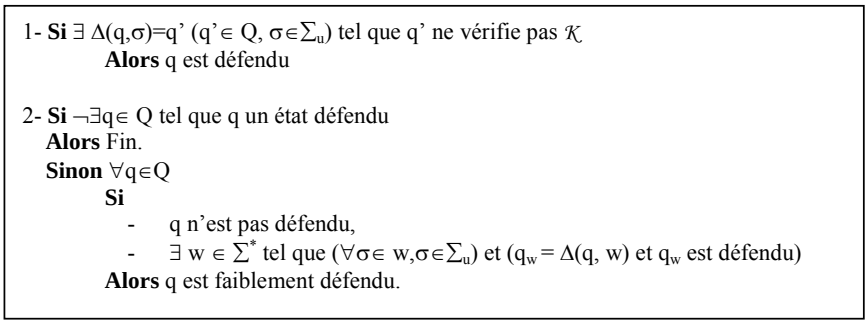


Figure 4.12.b. Traitement des événements non contrôlables (2ième étape).

La troisième étape 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.

$\forall q \in Q$ 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 $Q = Q - \{q'\}$

Figure 4.12.c. Retirer tous les états défendus et faiblement défendus (3ième étape).

3.2.3. Application sur un exemple simple.

Pour illustrer cet algorithme de synthèse, nous utilisons un exemple simple de mouvement vertical effectué par un vérin simple effet réalisant l'aspiration d'une pièce. Nous imposons à ce système, une contrainte de vivacité qui interdit le fait d'aspirer en position haute : **ASPIRER∧haut = 0**

Le comportement global du mouvement vertical et de mouvement d'aspiration est donné par un automate à états booléens de 24 états et 76 transitions (figure 4.13a). l'automate SUP obtenu comporte 14 états et 42 transitions. La figure 4.13b présente quelques évolutions de cet automate. En condition initiale, le système se trouve en position haute ($Z_0 = [0,0]$ et $E_0 = [1, 0,0]$).

Depuis l'état initial de l'automate du procédé, l'évolution $(p_0, \uparrow ASPIRER, p_1)$ est défendue car la contrainte n'est pas vérifiée. Par contre, l'évolution $(p_0, \uparrow DESCENDRE, p_2)$ n'est pas défendue.

L'occurrence de l'événement $\uparrow ASPIRER$ permet de quitter l'état p_{10} pour atteindre l'état p_{14}, cette évolution n'est pas défendue mais la séquence $\{\uparrow pièce \rightarrow \downarrow DESCENDRE \rightarrow \downarrow bas \rightarrow \uparrow haut\}$ permet d'atteindre l'état p_3 où la contrainte n'est pas vérifiée. L'état p_{17} est donc défendu (2ème étape de l'algorithme). De plus, l'occurrence de l'événement non contrôlable $\downarrow bas$ liant l'état p_{22} à l'état défendu p_{17} est faiblement défendu (2ème étape de l'algorithme). Après la détermination de ces états défendus et

faiblement défendus, l'algorithme supprime ces états ainsi que toutes les transitions liées à ces états.

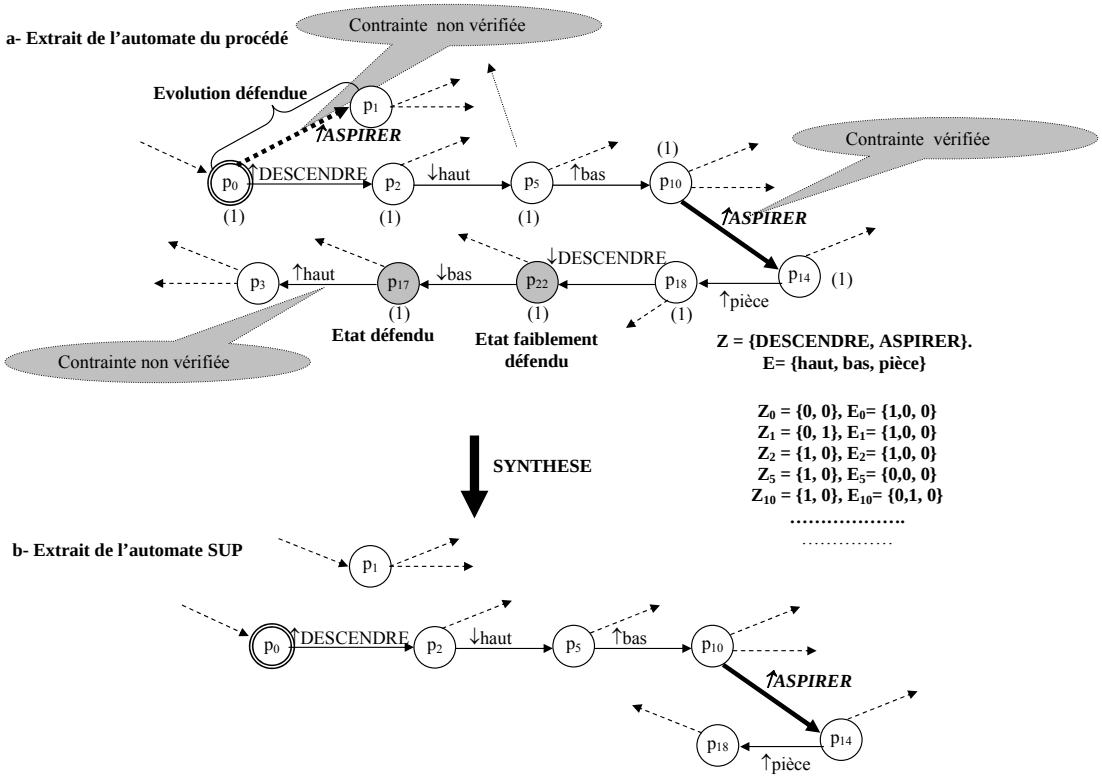


Figure 4.13. Exemple d'illustration.

3.3. Illustration sur l'exemple du préhenseur

Le tableau 4.6 présente l'ensemble des contraintes à respecter liées à chaque événement commandable.

Evénements commandable s	*Propriétés à respecter K_s*	*Contraintes modélisée*
↑AVANCER	$K_{\uparrow AVANCER}$= {❶❷❹}	- ❶ AVANCER ∧ RECULER =0 - ❷ AVANCER ∧ DESCENDRE =0 - ❸ RECULER ∧ DESCENDRE =0 - ❹ (AVANCER ∨ RECULER) ∧ ¬haut = 0 - ❺¬ASPIRER ∧ (bas ∧ gauche) = 0 - ❻ ASPIRER∧ (droite ∧ bas) = 0
↑RECULER	$K_{\uparrow RECULER}$= {❶❸❹}	
↑DESCENDRE	$K_{\uparrow DESCENDRE}$= {❷❸}	
↑ASPIRER	$K_{\uparrow ASPIRER}$= {❺}	
↓ASPIRER	$K_{\downarrow ASPIRER}$ = {❻}	

Tableau 4.6. Relation entre les événements commandables et les contraintes K_s.

Pour cet exemple, l'automate booléen décrivant le superviseur *SUP* est composé de 191 états et 736 transitions. Un extrait de cet automate est présenté dans la figure 4.14.b. Cet automate décrit la façon dont les différentes évolutions de l'automate de la figure 4.14.a sont désormais fixées afin de respecter les contraintes spécifiées. Par exemple, l'ordre *ASPIRER* peut survenir uniquement à partir de l'état p_{62}, après activation de *bas (contrainte ❺)*.

En fait, l'algorithme de synthèse du superviseur garantit qu'il en est de même pour toutes les évolutions de l'automate et pour l'ensemble des propriétés.

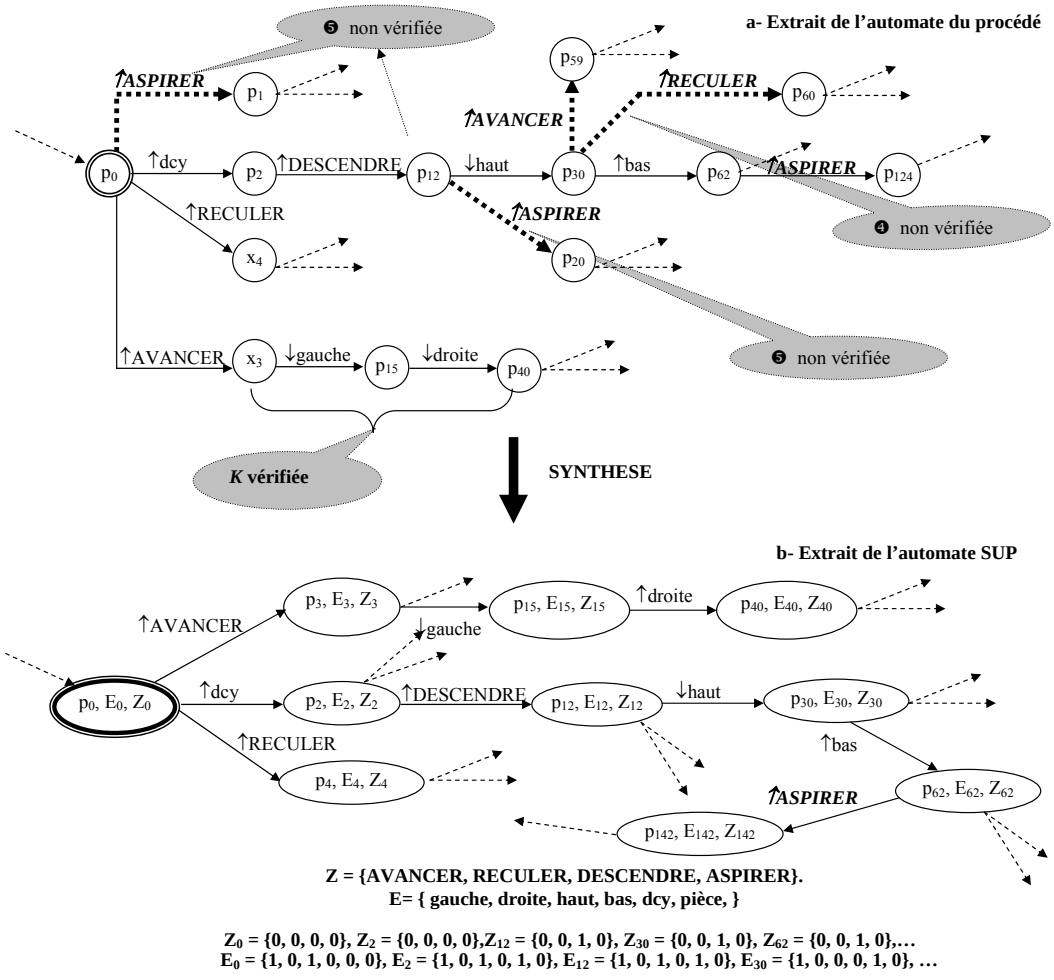


Figure 4.14 : Extrait de l'automate superviseur correspondant à l'exemple.

3.4. Complexité de calcul

Proposition :

Soit N supposé constant le nombre d'états dans chaque automate G_i de la PO avec i=1..n et k le nombre d'équations logiques modélisant les contraintes. La complexité de l'algorithme de synthèse est *O* ($k.N^n$).

Plus explicitement :

1. la construction de l'automate du procédé est de complexité N^n.

2. la construction des équations logiques représentant les contraintes est d'ordre k tel que k est le nombre de contraintes.

3. La construction du superviseur consiste, pour N^n états, à vérifier pour chaque état les évolutions possibles par rapport aux contraintes spécifiées. Dans le pire des cas, il faut vérifier k contraintes. La complexité de calcul pour chaque état est donc de l'ordre O(k).

➢ On peut dire alors que la complexité de notre approche de synthèse du superviseur est de l'ordre O ($k\ N^n$).

Discussion :

La génération d'un superviseur par l'algorithme de Kumar [KUM 91], passe par un produit croisé synchrone de la partie opérative et des contraintes. Ce produit conduit à une complexité de calcul de l'ordre O ($N^n.M^k$). A contrario, pour notre méthode de synthèse, la complexité est inférieure à celle de Kumar, elle est de l'ordre O ($k\ N^n$) (figure 4.15).

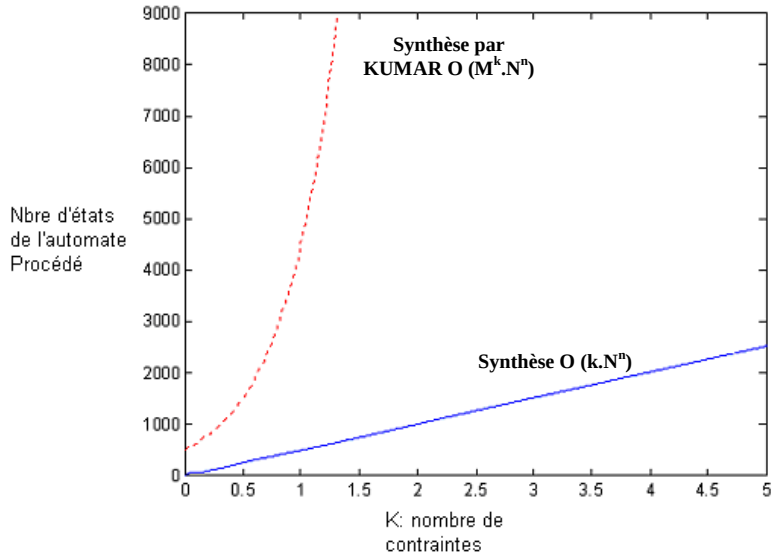


Figure 4.15. Comparaison entre les deux méthodes de synthèse.

Le tableau 4.7 montre sur l'exemple du préhenseur l'influence de la nouvelle démarche par rapport à l'ancienne, quant au nombre d'états calculés pour l'automate de supervision.

Dans le reste du document, nous utilisons cette nouvelle opération de synthèse avec les modèles automates à états booléens pour la partie opérative et les équations logiques pour les contraintes.

	Modélisation à base Automate à état fini obtenue de manière intuitive	Modèle de PO à base de règles + Modèle de contraintes à base d'équations logiques
Superviseur SUP	*1030 états*	191 états
	4143 *transitions*	736 transitions

Tableau 4.7: Influence de la modélisation des équations logiques sur la synthèse du superviseur.

Dans le reste du document, nous utilisons cette nouvelle opération de synthèse avec les modèles automates à états booléens pour la partie opérative et les équations logiques pour les contraintes.

4. SYNTHESE SOUS CONTROLE DU CONCEPTEUR : VERS L'INTEGRATION DU CONCEPTEUR DANS LA BOUCLE DE SYNTHESE

4.1. Introduction

Jusqu'à présent, les travaux que nous avons menés avaient pour objectif l'implantation de la commande en suivant une logique de synthèse automatique, c'est-à-dire en masquant au concepteur, les blocages traités et les corrections effectuées avant l'implantation. L'idée était de montrer les corrections sur le Grafcet originel au moment de l'exécution de la commande.

C'est au niveau de l'intersection entre l'automate de commande (ASS) et l'automate représentant le comportement commandable maximal admissible du procédé par rapport aux contraintes (*SUP*), qui se situe les informations sur les blocages en exécution réelle et sur les corrections (inhibition d'actions) apportées à la commande par la prise en compte des contraintes. C'est en exploitant les régions de *SYNC*, que l'étape suivante garantit que l'automate à implanter est non seulement déterministe et réactif mais il représente aussi la restriction minimale du comportement du Grafcet, garantissant la conformité par rapport aux contraintes spécifiées et le non-blocage du système.

Bien évidemment, cette démarche sécuritaire prend tout son sens lorsqu'il s'agit de systèmes complexes où il est illusoire de connaître de manière précise, pour chaque action et chaque situation de la commande, toutes les réactions du procédé et leur ordonnancement.

Pour des systèmes plus simples, la demande du concepteur est différente et s'oriente plutôt vers une demande d'aide à la conception de la commande. Les objectifs recherchés ne portent plus vraiment sur les notions d'optimalité de la commande et d'approche sécuritaire, mais vers la prise en compte du concepteur dans la boucle d'élaboration de la commande. En conséquence, de nouveaux services peuvent être proposés au concepteur comme par exemple la visualisation des séquences bloquantes qui sont totalement masquées dans la démarche automatique. Ceci permet au concepteur d'agir sur les raisons du blocage en modifiant par exemple la commande, en relâchant des contraintes ou en affinant le modèle de la partie opérative, de nouvelles itérations de

la synthèse étant alors effectuées à partir des modèles corrigés pour générer en finalité un modèle de commande correct [TAJ 03]. On parlera alors de démarche de synthèse de commande semi-automatique.

Par conséquent, il s'agit d'exploiter les informations de l'automate SYNC pour détecter (i) les blocages et (ii) les corrections introduites par la prise en compte des contraintes. La figure 4.16 illustre ces propos, deux cas pouvant se produire à l'issue de l'opération d'intersection :

- La commande respecte les spécifications imposées au système, alors le concepteur peut implanter la commande synthétisée dans un A.P.I. en toute sécurité, il est ainsi certain qu'il n'y a aucun risque pour le matériel opératif, les opérateurs ou le produit.
- La procédure d'intersection détecte soit :
  - Une situation de blocage. Le concepteur analyse la situation bloquante grâce aux informations élaborées à partir des régions de l'automate *SYNC* et agit en conséquence en modifiant par exemple la commande, en relâchant des contraintes ou en affinant le modèle de la partie opérative.
  - Le non-respect des intentions du concepteur. En effet, il existe dans l'automate *SYNC* des corrections générées pour prendre en compte les contraintes imposées au système. Le concepteur va visualiser une liste des contraintes non respectées au niveau de sa commande originelle et en déduire d'éventuelles modifications.

Une nouvelle itération de la procédure sera alors effectuée à partir du ou des modèles corrigés. Dans ce cas, la procédure de réduction est allégée et ne contient plus qu'une étape d'agrégation des situations instables et fournit en conséquence un automate implantable sur une cible quelconque.

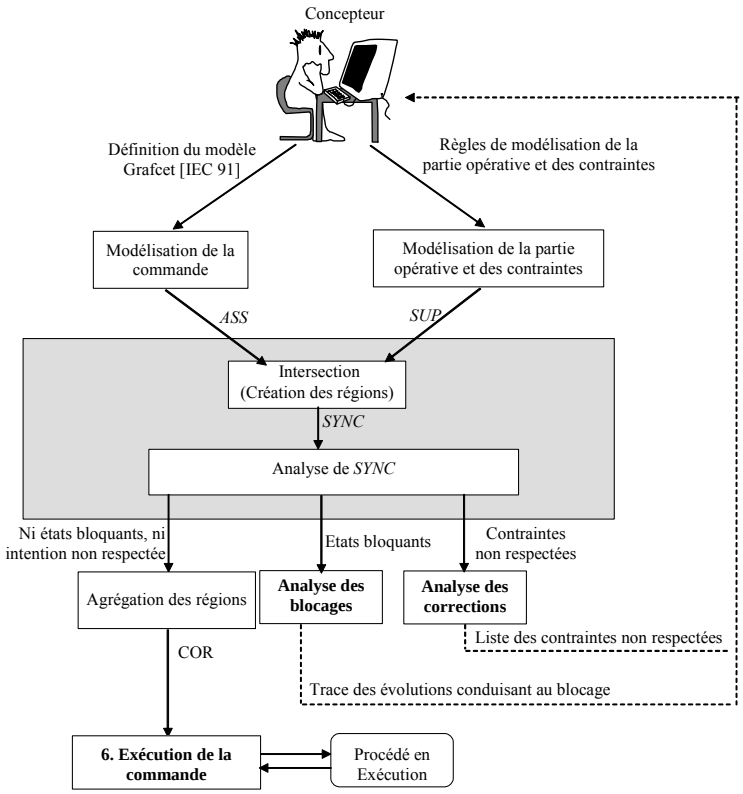


Figure 4.16 **:** Prise en compte du concepteur dans la boucle d'élaboration de la commande.

4.2. Visualisation et analyse des Séquences bloquantes

La visualisation des séquences bloquantes consiste à fournir au concepteur la trace d'évolution conduisant à l'état bloquant. Deux types de blocages doivent être distingués car la trace proposée ne sera pas la même. Il s'agit de blocages liés à l'occurrence d'un événement commandable et de ceux liés à un événement non commandable. Dans le 1er cas, la visualisation s'arrête à la région correspondante à l'état bloquant. En effet, cette unique région visualisée suffit pour donner l'information nécessaire sur la situation du Grafcet et les ordres envoyés dans cette situation à la partie opérative qui ont entraîné cette dernière dans une situation incompatible avec l'expression logique des transitions validées donc au blocage. Le deuxième cas concerne un blocage situé dans un état atteint par un événement non commandable. Dans ce cas, la visualisation s'arrête à la région caractérisée par une évolution commandable conduisant au blocage. En effet, une région correspond à une situation du Grafcet de spécification et permet de montrer le ou les derniers ordres envoyés de la commande vers la partie opérative et autorisés par le

superviseur et donc faciliter la détection de l'origine du blocage suite à l'envoi de cet ordre.

4.2.1. Algorithme de Visualisation

Pour visualiser les séquences bloquantes et proposer des éléments pertinents à analyser, il s'agit de construire la trace des évolutions sous forme d'un automate partiel défini à partir de l'automate d'intersection *SYNC*. Cet automate nommé *SEQ* est un 7-uplet SEQ = (Σ, ST_s, TR_s, st_b, r_a, ST_a, ST_{pb}) avec :

- $\Sigma = \Sigma_c \cup \Sigma_u$, où $\Sigma_c = \uparrow Z \cup \downarrow Z$ et $\Sigma_u = \uparrow E \cup \downarrow E$.

- ST_s est l'ensemble des états de SEQ partitionné en sous-ensembles appelés régions. $ST_s \subset ST$.

- TR_s: $ST_s \times \Sigma \rightarrow ST_s$ est une fonction partielle représentant les transitions de SEQ. $TR_s \subset TR$.

- $st_b \in ST_s$ est l'état initial de l'automate SEQ. Cet état correspond à un état bloquant de l'ensemble BLOC pour lequel est développé l'automate partiel SEQ.

- $r_a \subset ST$ est la région d'arrêt d'un pas de visualisation correspondant à la situation.

- $ST_a \subset ST$ contient l'ensemble des états amont des transitions entrantes à la région d'arrêt.

- $ST_{pb} \subset ST$ contient l'ensemble des états pseudo-bloquants conduisant à un état bloquant.

L'algorithme itératif pour un pas de visualisation est basé sur deux étapes (Figure. 4.18), il reçoit en entrée un automate SYNC = (Σ, ST, TR, st_0, r_0, BLOC) et la structure initiale de l'automate SEQ donnée par (Σ, $\{st_b\}$, $\varnothing$, st_b, $\varnothing$, $\varnothing$, $\{st_b\}$).

Pour chaque $st \in ST_{pb}$, le premier pas (❶) consiste à ajouter à l'automate *SEQ*, toutes les évolutions immédiatement précédentes conduisant à cet état.

Le traitement des états amont des transitions entrantes à l'état bloquant dépendant du type d'événements rencontrés, deux tests sont effectués dans le deuxième pas (❷) : Si la transition désigne un événement commandable (figure 4.17a), il faut ajouter à

l'automate *SEQ* la région englobant l'état bloquant et ajouter à l'ensemble ST_a, tous les états amont des transitions entrantes à cette région.

Si la transition représente un événement non commandable (figure 4.17b), il s'agit d'ajouter l'état amont de la transition entrante à l'ensemble des états pseudo bloquants ST_{pb} et de réitérer l'algorithme avec les éléments de cet ensemble. L'algorithme pour un pas de visualisation se termine lorsque la trace des évolutions contient la ou les régions possédant les actions qui devraient être interdites par la synthèse. Si l'utilisateur le souhaite, il peut élargir la visualisation en effectuant un autre pas à partir des éléments de l'ensemble ST_a.

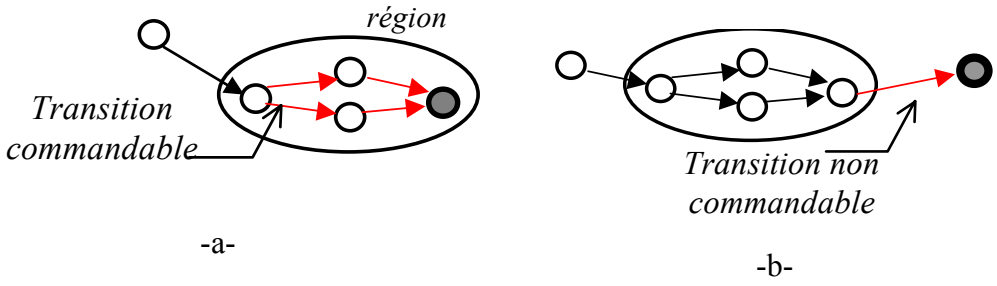


Figure 4.17. Principe de visualisation des séquences bloquante

$\forall$ st $\in ST_{pb}$

❶ $\forall$ (st', σ, st) $\in$ TR tel que $\sigma \in \sum$ Faire :
- (i) $ST_s = ST_s \cup \{st'\}$;
- (ii) $TR_s = TR_s \cup \{(st', \sigma, st)\}$;
- (iii) Afficher ST_s et TR_s ;

❷ $\forall$ (st', σ, st) $\in$ TR tel que $\sigma \in \sum$ Faire :

Si $\sigma \in \sum_c$ alors Faire :
- (i) r_a = region (st)
- (ii) $\forall$ st' $\in r_a$ et $\forall$ (st'', σ', st') $\in$ TR tel que $\sigma' \in \sum_u$ Faire :
  - Si (st'', σ', st') $\notin TR_s$ alors $TR_s = TR_s \cup$ (st'', σ', st') ;
  - Si st'' $\notin ST_a$ alors $ST_a = ST_a \cup \{st''\}$;
  - Afficher r_a, TR_s et ST_a ;

Si $\sigma \in \sum_u$ alors Faire :
- (i) Si st' $\notin ST_{pb}$ alors $ST_{pb} = ST_{pb} \cup \{st'\}$;
- (ii) $ST_{pb} = ST_{pb} - \{st\}$
- (iii) Afficher ST_{pb} ;

Figure 4.18. Algorithme d'un pas de visualisation d'une séquence bloquante.

4.2.2. Analyse des Séquences bloquantes

Pour faciliter l'analyse des séquences bloquantes visualisées, des informations nécessaires pour comprendre le phénomène du blocage et détecter son origine sont fournies au concepteur. Ces informations concernent les régions et les évolutions inter-régions (évolutions non commandables) conduisant au blocage. Il s'agit de :

- L'état du superviseur SUP et les évolutions commandables et non commandables autorisées dans cet état.
- La situation de l'automate des situations stables (ASS), les actions actives du Grafcet dans cette situation et les évolutions possibles sortantes de cette situation.
- Le vecteur d'entrée présentant l'état des capteurs.
- L'intersection entre les événements commandables autorisés et les actions actives dans la situation du Grafcet correspondante.

Cette liste d'informations est présentée au concepteur sous forme de tableaux. Chaque tableau caractérise deux aspects, le premier concerne la région visualisée (tableau 4.8) illustrant toutes les informations permettant sa construction et le deuxième tableau illustrant les transitions inter-règions (tableau 4.9).

SUP	**Evolution**					
	Etat amont	**Evénements commandables**		**Etat aval**		
	q_i	σ_i σ_{i+1}		q_{i+1} q_{i+2}		
ASS	**Situation Grafcet**	**Action du Grafcet**		**Transitions sortantes**		
	y_i	Actions		$f_i(e)$		
Région i du SYNC	**Intersection entre événements commandables et les actions**			**Région r_0**		
				Etat du SUP	**Etat de ASS**	**Vecteur d'entrée**
				q_{i}	**y_i**	**E_i**

Tableau 4.8 Région visualisée.

SUP	**Evolution**				
	Etat amont	**Evénements non commandables**	**Etat aval**		
	q_i	α_i	q_{i+1}		
ASS	**Situation Grafcet**	**Transitions sortantes**	**Situations atteintes**		
	y_i	$f_i(e)$ …	y_{i+1} …		
Région i SYNC	**Evolution inter-région**		**Etat atteint de la région atteinte r_{i+1}**		
			Etat de la PO	**Etat de ASS**	**Vecteur d'entrée**
			q_{i+1}	**y_{i+1}**	**E_i**

Tableau 4.9 Transition inter-région visualisée.

4.2.3. Illustration

Pour illustrer l'intérêt de la démarche de synthèse semi automatique, nous avons commis deux erreurs grossières : une dans la commande et autre dans la partie opérative.

4.2.3.1. Erreur dans la commande

La commande est donnée par le Grafcet de la figure 4.21 où l'ordre Reculer envoyé dans l'étape 8 et 9 du grafcet de commande a été remplacé par l'ordre AVANCER. La trace d'évolutions présentée au concepteur figure 4.19 montre qu'à partir de la région 25 aucune évolution n'est possible. C'est donc un blocage. L'analyse proposée au concepteur est ensuite donnée figure 4.20.

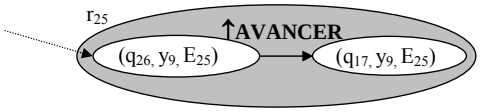


Figure 4.19. Etat bloquant suite à l'apparition d'un événement commandable

Superviseur	Evolution		
	Etat amont	Evénements commandables	Etat aval
	q_{26}	↑AVANCER ↑RECULER ↑DESCENDRE	q_{17} q_{34} q_{35}

ASS: automate des situations stables	Situation Grafcet	Actions du Grafcet	Transition sortante
	y_9	AVANCER	↑gauche./haut

Région 25 de l' Automate SYNC	Intersection entre événement commandable et les actions	Région r_{25}	
		Etat amont	Etat aval
	↑AVANCER	(q_{26}, y_9, E_{25}) E_{25}= [0 1 1 0 1 0]	(q_{17}, y_9, E_{25}) E_{25}= [0 1 1 0 1 0]

région 25 (blocage)		
$(q_{26}\ y_9\ E_{25})$	↑AVANCER	$(q_{17}\ y_9\ E_{25})$

E= [gauche, droite, haut, bas, dcy, pièce].

Figure 4.20 Analyse de la séquence bloquante liée à une erreur au niveau de la commande.

L'analyse de la séquence de blocage donnée sous forme de tableaux procure au concepteur les informations suivantes :

- Dans la situation bloquante, le Grafcet se trouve dans l'étape 8 du Grafcet où l'ordre AVANCER doit être envoyé vers la partie opérative
- Le superviseur autorise l'événement *↑AVANCER*.
- La PO se situe à droite et en haut

- L'intersection entre les ordres actifs dans la situation y_9 de *ASS* et les ordres autorisés par le superviseur est $\uparrow$*AVANCER* ce qui signifie que *AVANCER* est envoyé vers la partie opérative.
- Pour quitter la situation y_9, il faut avoir gauche. /haut.

A partir de ces informations et du modèle de la partie opérative associée à l'ordre *AVANCER* (mouvement horizontal (figure 4.21) le concepteur peut constater que si cet ordre est reçu au niveau de la partie opérative, ce sont les événements non commandables $\downarrow$*gauche* puis $\uparrow$*droite* qui vont être envoyés à la partie commande, ce qui est en contradiction avec l'information, selon laquelle, pour quitter la situation y_9, il faut *gauche. /haut*. Le concepteur en déduit donc qu'il y'a un problème quant à l'envoi de l'ordre *AVANCER*.

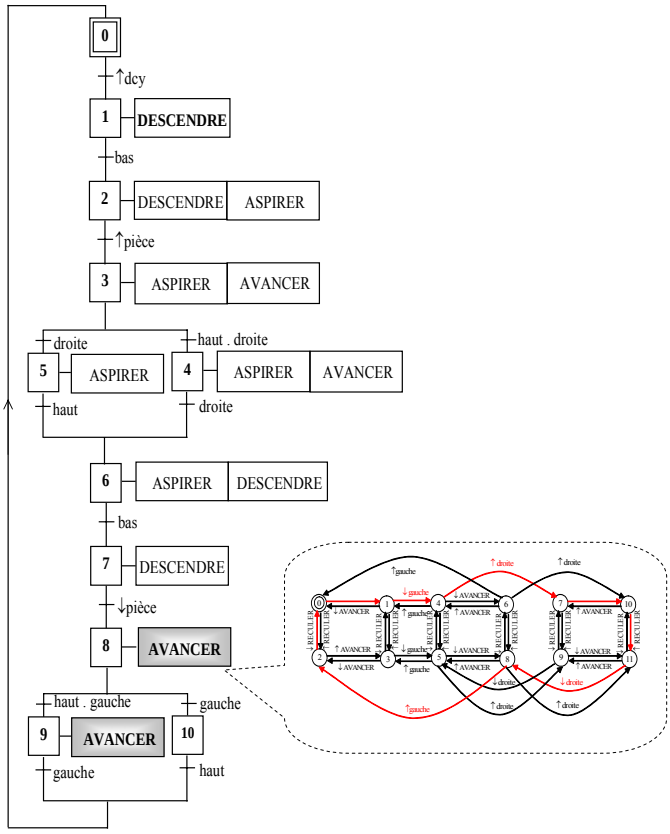


Figure 4.21. Grafcet originel et modèle du mouvement horizontal.

4.2.3.2. Erreur au niveau de la partie opérative

Dans cette partie, l'erreur commise concerne le mouvement vertical où nous avons substitué entre les états 2 et 3 du modèle, l'événement $\uparrow$*bas* par $\uparrow$*haut* (figure.4.22).

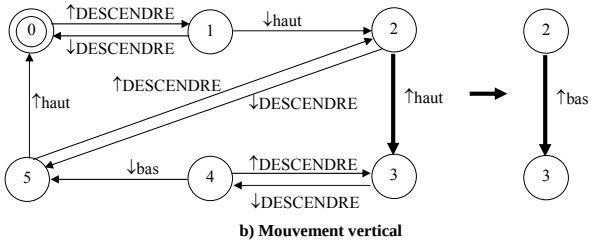


Figure 4.22. Modèle du mouvement vertical.

Dans ce cas de figure, la région r_{19} est bloquante car aucune évolution n'est possible à partir de cette région. C'est un événement non commandable qui conduit à cette région, il faut donc trouver l'origine. Par conséquent, la trace proposée au concepteur inclut les régions *r_5, r_{13} et r_{19}* (figure 4.23). L'analyse proposée au concepteur est ensuite donnée figure 4.24.

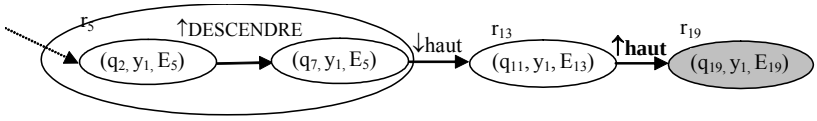


Figure 4.23. Etat bloquant suite à l'apparition d'un événement non commandable.

L'analyse proposée au concepteur est ensuite donnée figure 4.24.

L'analyse de la séquence de blocage donnée sous forme de tableaux procure au concepteur les informations suivantes :

- Envoi de l'ordre DESCENDRE dans la région r_5 car il est autorisé par le superviseur et actif dans l'étape 2 (y_2 de ASS) du grafcet.
- Evolution vers la région *r_{13}* sur occurrence de l'événement non commandable *↓haut* car l'événement issu de la partie opérative suite à l'ordre *DESCENDRE* est également *↓haut (q_2, ↓haut, q_{11})*.
- Dans la région *r_{13}* pas d'ordre envoyé (maintien de l'activation de *DESCENDRE*) sachant que la commande se situe toujours à l'étape 2.
- Evolution vers la région *r_{19}* sur occurrence de l'événement non commandable *↑haut (q_{11}, ↑haut, q_{19})*.
- Région *r_{19}* pas d'ordre envoyé (maintien de *DESCENDRE*) et la commande se situe toujours à l'étape 2.

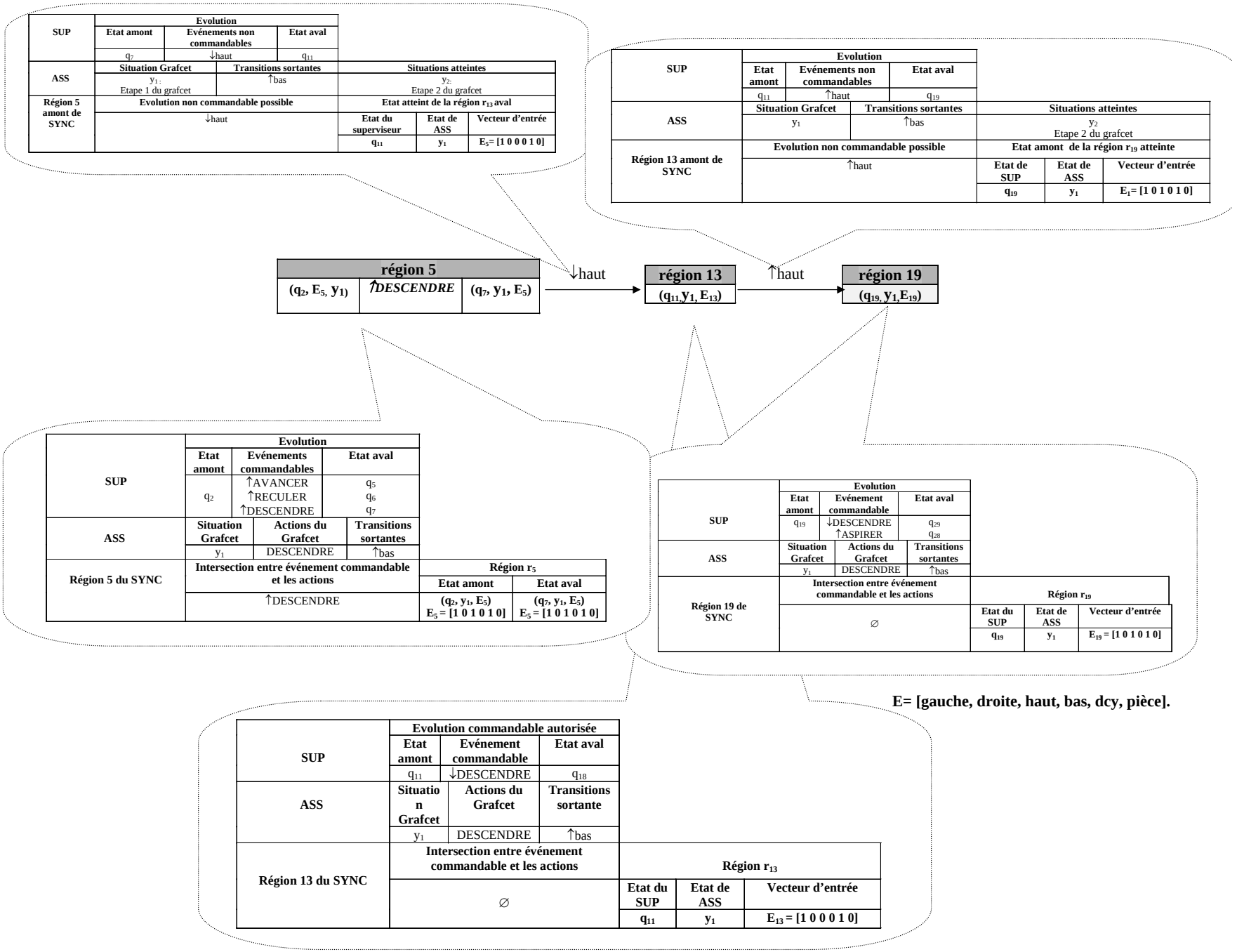


Figure 4.24 : Analyse du concepteur.

- Le vecteur d'entrée dans l'état bloquant indique que haut =1.
- Pour quitter la situation y_2 de *ASS*, il faut valider l'expression $\uparrow bas$ pour arriver à la situation y_1.

A partir de ces informations, le concepteur constate facilement que l'erreur est liée à l'occurrence de l'événement $\uparrow haut$ de la partie opérative qui empêche la validation de l'expression logique $\uparrow bas$ issue de ASS, ce qui conduit au blocage. Il en déduit alors qu'il ne peut s'agir d'une erreur de commande mais d'une erreur au niveau du modèle de la partie opérative.

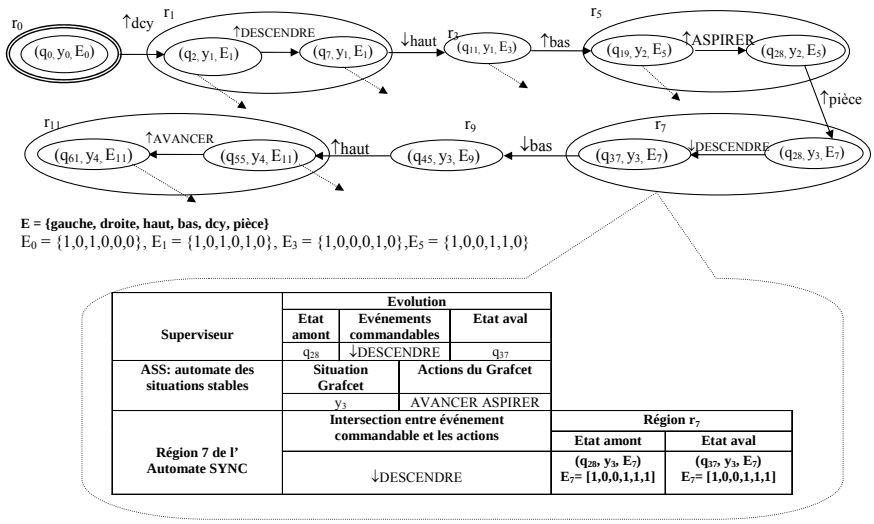
4.3. Visualisation des contraintes non respectées par la commande originelle

La visualisation d'une liste des contraintes non respectées au niveau de la commande originelle consiste à remonter de l'automate SYNC toutes les informations possibles concernant les évolutions autorisées et supprimées par l'étape d'intersection. En effet, elle consiste à utiliser les mêmes tableaux employés pour visualiser les séquences bloquantes mais cette fois pour montrer les évolutions filtrées par l'étape d'intersection.

Pour l'exemple du préhenseur le résultat de la procédure d'intersection montre que l'intention originelle du concepteur dans la modélisation de la commande n'est pas respectée dans certaines situations de la commande. Les tableaux mis à disposition du concepteur lui permettent alors d'analyser les situations et de visualiser une inhibition d'ordres qui ne correspond pas à l'intention du concepteur.

L'intersection de ASS et de SUP conduit à l'obtention de l'automate de la figure 4.25 où le concepteur détecte que dans la région r_7, le seul événement envoyé vers la partie opérative est $\downarrow DESCENDRE$. L'ordre *AVANCER* qui est présent au niveau de ASS et interdit par le superviseur n'est donc pas envoyé dans cette région. En observant de plus près les évolutions entre la région 7 et la région 11, le concepteur remarque que la partie opérative doit atteindre la position haute avant de recevoir l'ordre AVANCER. Cette inhibition d'ordre est liée à la prise en compte de la contrainte « *Ne pas Avancer ou reculer sans être en position haute* ». En conséquence, cela signifie que le grafcet proposé par le concepteur ne prend pas en compte la contrainte de fonctionnement (étape 3 du Grafcet de la figure 4.26).

Soit le concepteur doit effectuer une correction de la commande en ajoutant une condition permettant de vérifier que la position haute est atteinte avant de donner l'ordre *AVANCER*, soit il laisse la synthèse traiter automatiquement la correction.



Superviseur	Evolution		
	Etat amont	Evénements commandables	Etat aval
	q_{28}	↓DESCENDRE	q_{37}

ASS: automate des situations stables	Situation Grafcet	Actions du Grafcet
	y_3	AVANCER ASPIRER

Région 7 de l' Automate SYNC	Intersection entre événement commandable et les actions	Région r_7	
		Etat amont	Etat aval
	↓DESCENDRE	**(q_{28}, y_3, E_7) E_7= [1,0,0,1,1,1]**	**(q_{37}, y_3, E_7) E_7= [1,0,0,1,1,1]**

Figure 4.25. Analyse du concepteur des corrections apportées par les contraintes.

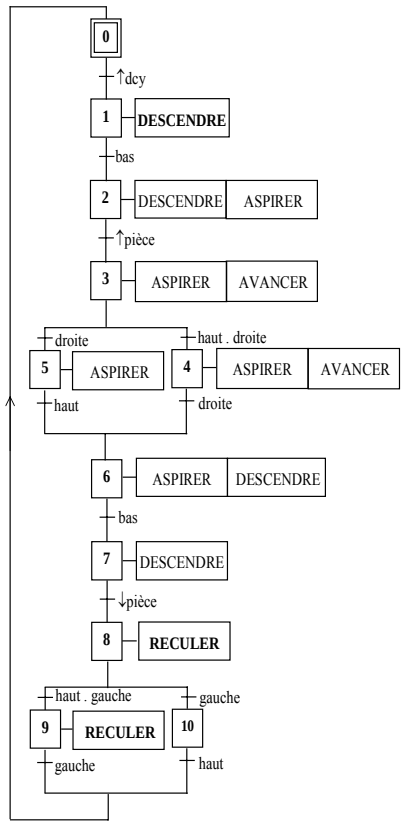


Figure 4.26. Grafcet de commande proposé par le concepteur.

5. OUTIL DE SYNTHESE DE COMMANDE

5.1. OPTIGRAF7 : Réalisation de la synthèse de commande

Les travaux réalisés ont entraîné la conception et le développement d'un outil informatique dédié *OPTIGRAF7*, présenté figure 4.27. Développé sur PC, cet outil est composé d'un environnement *d'Edition et l'autre de Calcul* permettant l'édition des modèles de spécification et la génération de la commande optimale. Le premier environnement intègre un éditeur de Grafcet, un éditeur d'automates, l'outil AGGLAE [ROU 94] pour exécuter la phase d'extraction (étape 3) et des modules développés en C++ (figure 4.28) pour effectuer les phases de synthèse, d'intersection, de réduction et de visualisation (étapes 2, 4 et 5). L'élaboration de l'automate de commande optimale peut s'effectuer « hors ligne » ou « en ligne ».

Cet outil est adapté aux différentes extensions développées dans le cadre de cette thèse en intégrant la partie visualisation des séquences bloquantes et le calcul du superviseur en utilisant l'automate à états booléens pour la partie opérative et les équations logiques pour les contraintes.

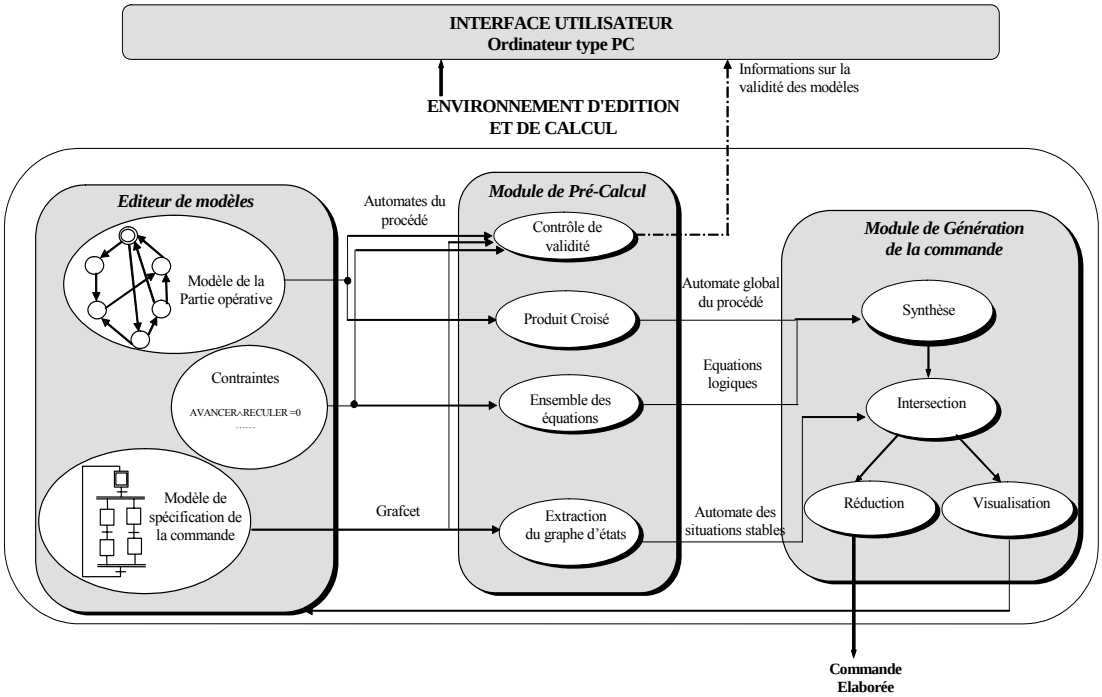


Figure 4.27. Structure d'OPTIGRAF7.

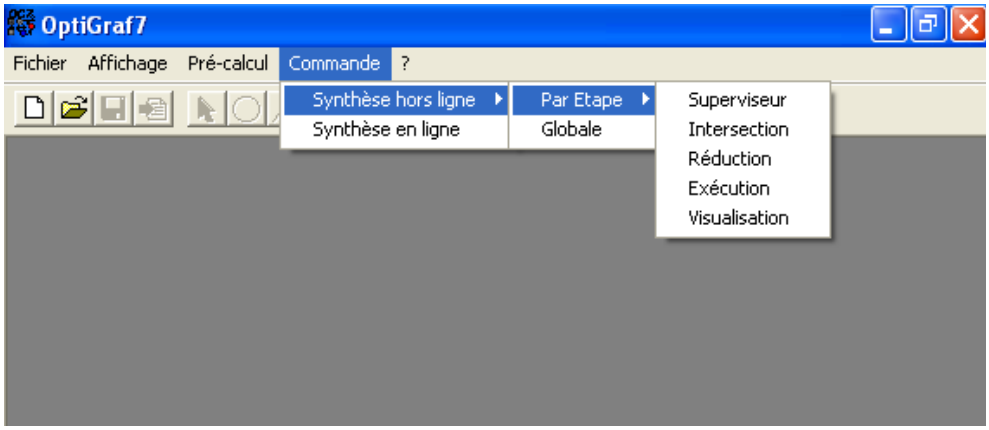


Figure 4.28. Barre de menus d'optigraf7.

5.2. Implantation

L'automate de commande ainsi construit peut être exprimé dans plusieurs langages normalisés pour les A.P.I. [IEC 93]. Nous avons retenu pour notre part, le langage IL (Liste d'Instruction), le langage ST (Structured Text) et le langage Ladder (LD), la traduction de l'automate de commande est dans la figure 4.29. Le programme se compose de deux parties : le traitement des évolutions et l'affectation des sorties. Le traitement des évolutions s'effectue par l'intermédiaire de contacts qui activent les états correspondants, ceux-ci conditionnant ensuite l'affectation des sorties.

Cette implantation présente plusieurs avantages : les trois langages sont des langages de programmation normalisés et ils sont les plus utilisés dans les équipements de contrôle/commande industriels. Les primitives « Set » et « Reset » permettent de traduire facilement l'automate de commande *COR*. Il existe cependant un inconvénient majeur de l'implantation dans un API qui est lié au temps de cycle du programme A.P.I. En effet, le temps de cycle dépend directement du nombre d'instructions et peut nuire à la réactivité de la commande élaborée. Il est possible de s'affranchir du temps de cycle inhérent à tout A.P.I. en utilisant une architecture générique qui combine la notion de consultation de tables et la notion de tâche sous interruption [NDJ 99]. Cependant, le manque de normalisation de cette architecture est un frein à son utilisation.

C'est pourquoi, afin de pouvoir tester nos propositions en terme de génération de l'automate de commande par Optigraf7, un module de traduction à été développé spécifiquement. Ce dernier traite les fichiers générés par l'outil de synthèse Optigraf7, et les transforme, en fonction de nos méthodes, en un fichier utilisable dans l'automate programmable Télémécanique TSX (figure 4.30). Ainsi il est possible de simuler le

comportement de la commande obtenue par rapport à celle de départ, mais également de générer des éléments de code automate téléchargeable dans les automates TSX 37-21.

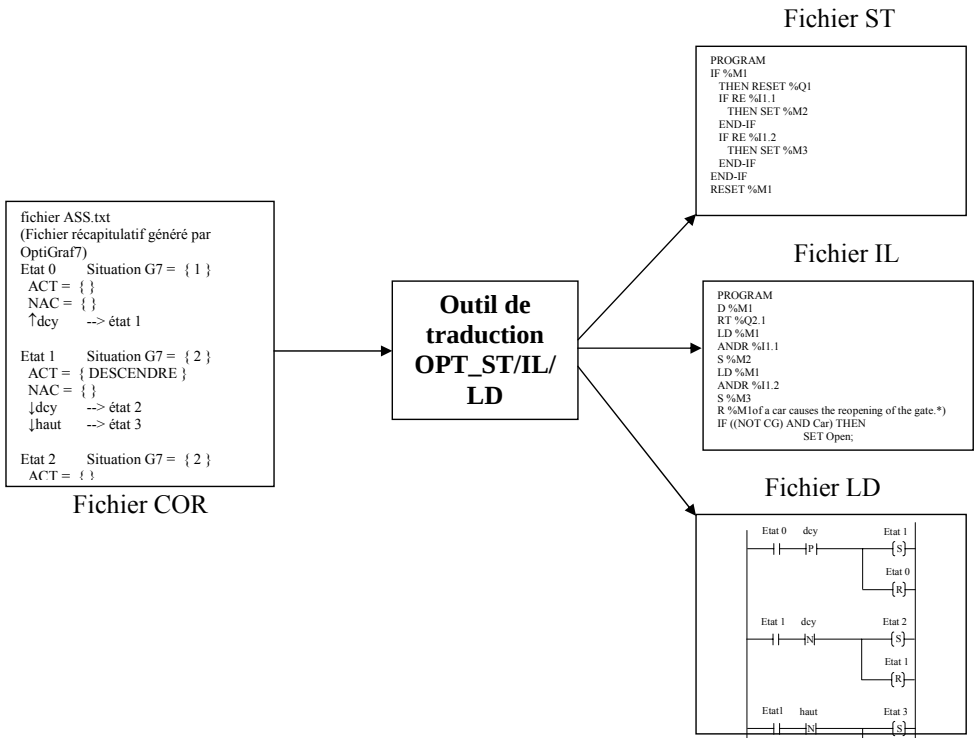


Figure 4.29. Principe du fonctionnement de OPT_ST/IL/LD.

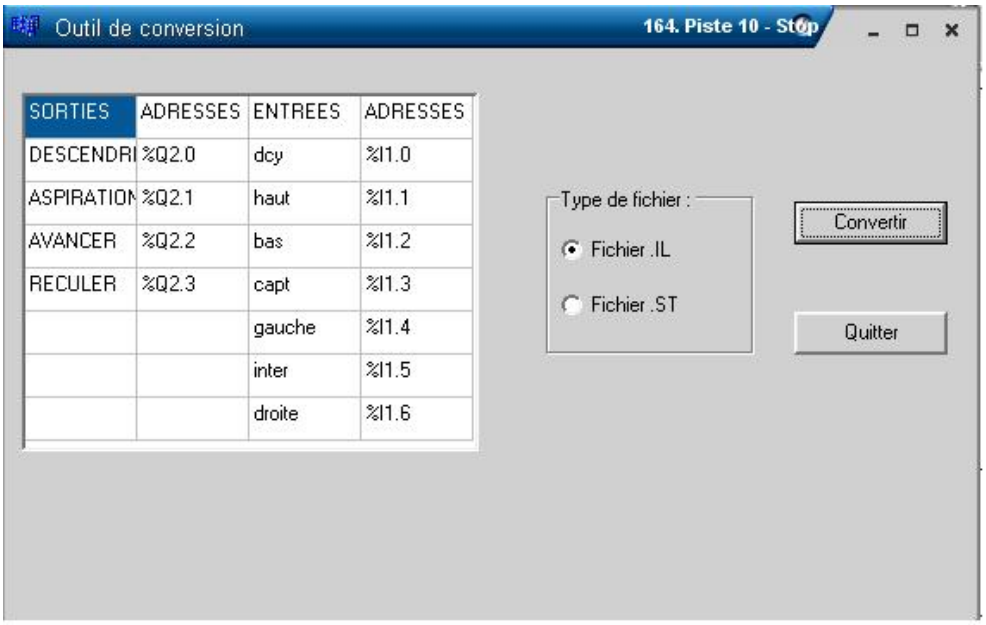


Figure 4.30. Fenêtre de conversion de COR en un langage **API *TSX 37-22.***

6. CONCLUSION

Nous avons donc présenté dans ce chapitre une extension de la démarche de synthèse d'une commande sûre, réactive, déterministe et sans blocage spécifiée par Grafcet. Celle-ci concerne la prise en compte d'une modélisation évoluée de la partie opérative et des contraintes.

Dans un premier temps, nous avons appliqué la démarche intuitive et progressive (section 2 du chapitre 2) en utilisant ces nouveaux modèles permettant de passer de la spécification à l'implantation de la commande. Cette mise en œuvre a montré l'intérêt de modéliser les contraintes par des équations logiques en terme de réduction de l'explosion combinatoire par rapport à l'utilisation des automates à états. En plus, elle montre l'intérêt d'intégrer un modèle de la PO en appliquant une opération d'intersection entre l'automate des situations stables contraint ASSS et la PO. Cependant cette démarche a des limites concernant la non prise en compte de toutes les contraintes de sûreté et de vivacité, en se basant seulement sur les contraintes liées aux ordres actifs et non celles liées aux ordres non actifs.

Dans un deuxième temps, nous avons repris la démarche en donnant cette fois un cadre formel, plus solide, à notre approche et en étendant le processus de synthèse, selon RW. Cette extension utilise la théorie de supervision selon RW [RAM 89] en adaptant l'algorithme de synthèse à la nouvelle modélisation (un automate à états booléens obtenu par une structuration à base de règles pour la PO et par équations logiques dans l'algèbre de Boole pour les contraintes).

Enfin, nous avons proposé dans ce chapitre une méthode de synthèse sous contrôle du concepteur dans le but d'influencer les modèles de départ et de relancer une nouvelle vérification formelle par synthèse, jusqu'à obtenir un modèle de commande déterministe, réactif, sûr et sans blocage. Cette méthode permet ainsi de raffiner la modélisation du Grafcet, les contraintes et de la partie opérative, et de traiter les blocages de façon non plus automatique mais semi-automatique, en intégrant le concepteur dans la boucle de synthèse.

Chapitre 5

APPLICATION :

SYSTEME DE TRI DE CAISSES

L'objectif de ce chapitre est d'illustrer notre contribution à travers la conception d'une commande d'un procédé réel (système de Tri de caisses de complexité moyenne. Il s'agit d'utiliser toutes les argumentations données aux chapitres 3 et 4 concernant la synthèse de commande. Ce chapitre illustre aussi l'intégration du concepteur dans la boucle de synthèse qui permet d'éliminer les blocages pouvant surgir dans l'étape de synthèse. Enfin, une comparaison chiffrée est élaborée pour illustrer l'amélioration apportée par cette contribution présentée dans ce mémoire par rapport à la méthode de synthèse présentée initialement.

1 PRESENTATION ET CAHIER DES CHARGES

Le procédé réel sur lequel nous avons appliqué notre démarche, est un dispositif automatique destiné à trier des caisses de deux tailles différentes et se compose d'un tapis amenant les caisses (tapis 1), de quatre poussoirs (P1, P2, P3 et P4) et de deux tapis d'évacuation (tapis 2 et tapis 3), comme illustré figure 5.1.

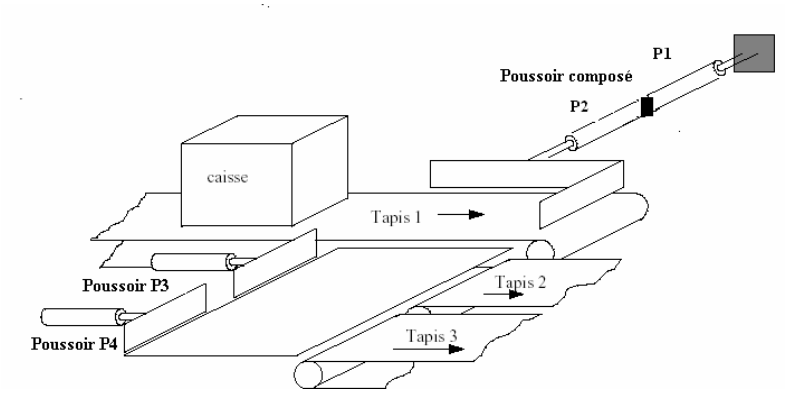


Figure 5.1. Système de Tri de caisses.

Dans cette application, nous utilisons pour pousser les pièces devant les poussoirs P3 et P4 un poussoir composé de deux vérins double effet P1 et P2 montés dos à dos, assemblés par un support permettant d'obtenir trois positions différentes (figure 5.2). Pour les autres poussoirs (P3 et P4), nous utilisons des vérins simples effets.

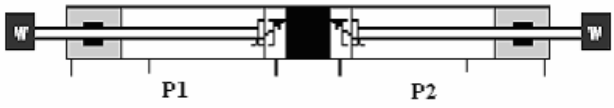
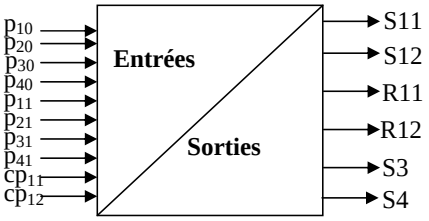


Figure 5.2. Technologie du poussoir composé.

Le système est défini par dix entrées et six sorties. Les entrées sont composées de :



Entrées/sorties du système.

- p_{10} : fin de course d'entrée du vérin P1
- p_{20} : fin de course d'entrée du vérin P2

- p_{30} : fin de course d'entrée du vérin P3
- p_{40} : fin de course d'entrée du vérin P4
- p_{11} : fin de course de sortie du vérin P1
- p_{21} : fin de course de sortie du vérin P2
- p_{31} : fin de course de sortie du vérin P3
- p_{41} : fin de course de sortie du vérin P4
- cp_{11} : capteur de présence de la petite caisse
- cp_{12} : capteur de présence de la grande caisse

Les sorties sont définies par :

- S11 : ordre de sortie du vérin P1
- S12 : ordre de sortie du vérin P2
- R11 : ordre de recul du vérin P1
- R12 : ordre de recul du vérin P2
- S3 : ordre de sortie du vérin simple effet P3
- S4 : ordre de sortie du vérin simple effet P4

Le tapis 1 amène les unes derrière les autres et dans un ordre quelconque, deux types de caisses caractérisées par deux tailles différentes. Les caisses de petite taille sont définies par cp_{11} et les autres de grande taille sont définies par cp_{12}. Le poussoir composé place ces caisses devant l'un des tapis d'évacuation (tapis 1 ou tapis 2), selon leurs types. Les petites caisses sont placée devant le tapis 2 par la sortie du vérin P1, par contre, les grandes caisses sont placées devant le tapis 3 par la sortie successive des vérins P1 puis P2.

Le retour du poussoir composé s'effectue dès que l'une des deux caisses est présente devant l'un des deux tapis 2 ou 3. Après cela, dans le cas d'une petite caisse, le vérin P3 la pousse sur le tapis 2, par contre, dans le cas d'une grande caisse, le vérin P4 la pousse sur le tapis 3 pour évacuation.

Dans la situation initiale, tous les éléments du système sont immobiles et tous les vérins sont rentrés (p_{10}, p_{20}, p_{30}, p_{40}).

2 MODELISATION

L'étape de modélisation représente une étape de base très délicate dans la démarche de synthèse. Elle consiste donc à spécifier la commande par Grafcet, les contraintes à respecter par le procédé et à modéliser le comportement de la partie opérative.

2.1 Grafcets de spécification

Afin de réaliser l'objectif de l'automatisation, une modélisation de la commande à base de trois Grafcets partiels, G_1, G_2 et G_3 est donnée figure 5.3. Ceux-ci décrivent respectivement les commandes du déplacement du poussoir 1 (vérin P1 et vérin P2), celui du poussoir P3 ainsi que celui du poussoir P4.

Ces Grafcets sont synchronisés par le biais des conditions $X_{111}.p_{11}.p_{20}$ et $X_{102}.p_{21}.p_{11}$ correspondantes à la présence de la caisse devant l'un des deux poussoirs P3 ou P4, et les variables internes X300 et X400 indiquant la fin de dépôt de la caisse sur les tapis 2 et/ ou 3. Chaque Grafcet partiel revient à son étape initiale dés lors que le cycle est terminé, indépendamment des évolutions des autres Grafcets.

Description du Grafcet G1 : l'arrivée d'une petite caisse déclenche la sortie du vérin P1 vers la position intermédiaire $p_{11,}$ c'est-à-dire devant le tapis 2, suite à l'activation de l'étape X111, donc l'activation de l'ordre *S11*. Dès que la caisse se situe devant le tapis 2 (devant le poussoir P3), le franchissement de la $p_{20}.p_{11}$ conduit à l'étape X111 pendant laquelle le retrait du vérin est actionné (*R11*) jusqu'à la fin de course p_{10}. De la même façon, l'arrivée d'une grande pièce déclenche le mouvement de P1 (*S11*) vers la position intermédiaire p_{11}, puis la sortie du poussoir P2 (*S12*) vers la position maximale ($p_{11}.p_{21}$). La commande se trouve dans l'étape 103 et le retrait du poussoir 1 s'effectue successivement par le retrait du vérin P2 puis le vérin P1. La caisse est alors déposée sur le tapis 3 et un nouveau cycle peut reprendre si toutefois les caisses ont bien été évacuées.

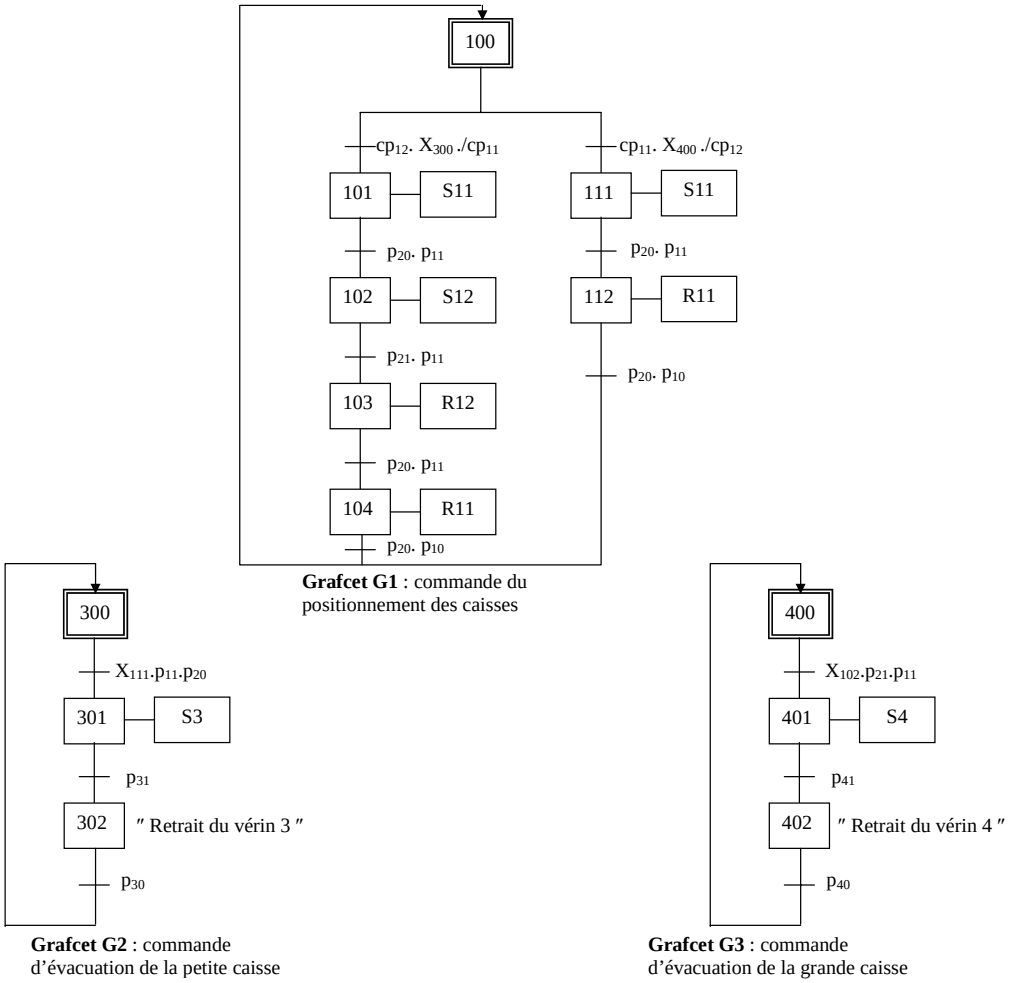


Figure 5.3. Grafcets de spécification.

Description du Grafcet G2 : Dès que la petite caisse (cp_{11}) est située devant le poussoir 3 indiqué par la variable interne X_{111}, le Grafcet se trouve dans l'étape X_{301} correspondante à la sortie du poussoir 3. Le franchissement de la transition conditionnée par le capteur p_{31} provoque l'activation de l'étape X_{302} permettant la désactivation de l'action *S3* conduisant au recul du vérin P3.

Description du Grafcet G3 : De la même manière que le Grafcet G2, ce Grafcet représente la commande du poussoir 4 qui permet d'évacuer la grande caisse sur le tapis 3 dés que cette dernière est positionnée devant le vérin.

2.2 Spécification et modélisation des contraintes

Durant le fonctionnement du système, 8 contraintes sont définies :

i) interdiction de pousser les vérins P3 et P4 en même temps

ii) Ne pas pousser le vérin P3 et P1 en même temps

iii) Ne pas sortir et rentrer le vérin P1 en même temps.

iv) Ne pas sortir et rentrer le vérin P2 en même temps.

v) Interdiction de pousser le vérin P2 si la petite caisse est détectée

vi) Ne pousser le vérin P4 qu'en présence de la grande caisse

vii) Ne pas sortir le vérin P3 si le vérin P1 est en position sortie p_{11}.

viii) Ne pas sortir le vérin P4 si le vérin P2 est en position sortie p_{21}.

Pour le système de tri de caisse, nous imposons huit contraintes de sécurité. Les deux premières contraintes consistent à interdire l'activation simultanée des ordres *S11* et *S3*, et des ordres *S3* et *S4*. Les équations logiques représentant ces contraintes sont données par :

- ❶ $\neg(S11 \wedge S3)$
- ❷ $\neg(S3 \wedge S4)$

Les deux contraintes « ne pas avoir *S11* et *R11* activées en même temps », et « les deux ordres S12 et R12 activées en même temps » se traduisent par :

- ❸ $\neg(S11 \wedge R11)$
- ❹ $\neg(S12 \wedge R12)$

La cinquième contrainte de sécurité étudiée n'autorise pas le mouvement du vérin 2 en présence de la petite pièce et s'exprime par l'équation logique suivante.

- ❺ $\neg(S12 \wedge cp_{11})$

La quatrième contrainte de sécurité étudiée n'autorise le mouvement du vérin 4 qu'en présence de la grande pièce et s'exprime par l'équation logique suivante.

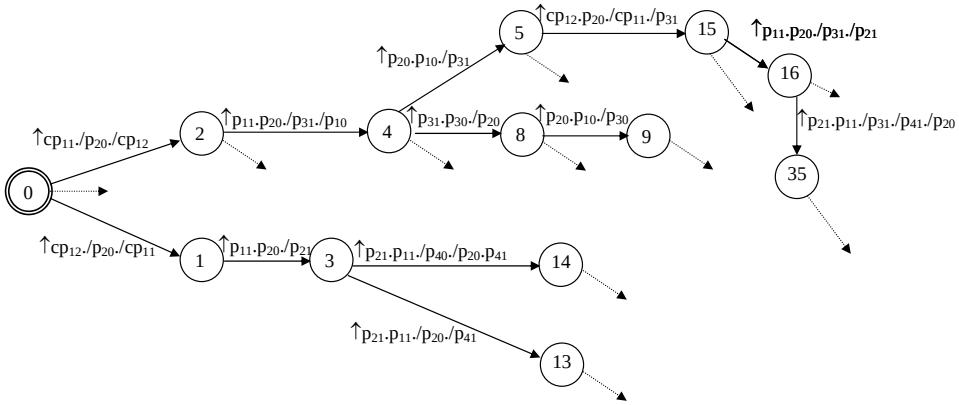
- ❻ $\neg(S4 \wedge \neg cp_{12})$

Deux autres contraintes de vivacité sont prises en considération. Elles concernent l'interdiction d'évacuation de la petite ou la grande pièce tant que les vérins de positionnement P1 et P2 ne sont complètement pas rentrés.

- ❼ $\neg(S3 \wedge p_{11})$
- ❽ $\neg(S4 \wedge p_{12})$

2.3 Analyse de la commande

La figure 5.4 présente les premières évolutions de la commande spécifiée par les Grafcets de la figure 5.3. Le modèle global obtenu par l'outil AGGLAE comporte 39 situations et 342 transitions. Une analyse rapide de ce modèle montre, par exemple, que l'interdiction de la sortie du vérin P3 et de la sortie de P1 en même temps, n'est pas respectée. En effet, à partir de la situation 5, l'arrivée d'une grande caisse cp_{12} sur le tapis, fait évoluer la commande vers la situation 15, correspondante à l'activation des étapes 101, 400 et 301 du Grafcet. Dans cette situation, les ordres *S3* et *S11* sont simultanément actifs, ce qui contredit la contrainte de sécurité que nous allons fixer par la suite. Une prise en compte d'une démarche de synthèse doit être appliquée pour éliminer les situations non conformes aux contraintes spécifiées et à toutes les évolutions irréelles.



Etapes$_{sit0}$= {100, 400, 300} ; Z$_{sit0}$= ∅ ;
Etapes$_{sit1}$= {101, 400, 300} ; Z$_{sit1}$= {S11} ;
Etapes$_{sit2}$= {111, 400, 300} ; Z$_{sit2}$= {S11} ;
Etapes$_{sit3}$= {102, 400, 300} ; Z$_{sit3}$= {S12} ;
Etapes$_{sit4}$= {112, 400, 301} ; Z$_{sit4}$= {R11, S3} ;
Etapes$_{sit5}$= {100, 400, 301} ; Z$_{sit5}$= {S3} ;
Etapes$_{sit8}$= {112, 400, 302} ; Z$_{sit8}$= {R11} ;
Etapes$_{sit9}$= {100, 400, 302} ; Z$_{sit9}$= {} ;
Etapes$_{sit13}$= {103, 401, 300} ; Z$_{sit13}$= {R12, S4} ;
Etapes$_{sit14}$= {103, 402, 300} ; Z$_{sit14}$= { R12} ;
Etapes$_{sit15}$= {101, 400, 301} ; Z$_{sit15}$= {S11, S3} ;
Etapes$_{sit16}$= {102, 400, 301} ; Z$_{sit16}$= {S12, S3} ;
Etapes$_{sit35}$= {103, 401, 301} ; Z$_{sit35}$={R12, S3, S4} ;

Figure 5.4. Quelques évolutions de l'automate ASS.

Nous avons montré précédemment qu'un modèle de la commande optimale théoriquement sans blocage, peut présenter des situations qui peuvent conduire à des blocages en exécution réelle. Ce type de blocage est engendré lorsque l'état de la partie opérative, est en contradiction avec les réceptivités des transitions validées du Grafcet, interdisant ainsi toute évolution du système ou lorsque l'ordre, ou les ordres envoyés à la PO peut provoquer un non respect des contraintes de sûreté de fonctionnement.

La démarche de synthèse sur le système de tri de caisses, nous permet de mettre en évidence de tels blocages, par exemple celui illustré par l'évolution successive de la

commande de la situation 5 à la situation 15, puis à la situation 16 (figure 5.4). De la situation 5 correspondante à la sortie du vérin P3 pour l'évacuation de la petite caisse, le poussoir composé est en position rentrée. La transition sortante de la situation 5 caractérise la détection d'une grande caisse *(cp_{12})*, ce qui conduit vers la situation 15 où les deux ordres *S3* et *S11* sont envoyés vers la partie opérative. La transition $\uparrow p_{11}.p_{20}./p_{31}./p_{21}$ correspondante à la sortie du vérin P1 permet d'atteindre la situation 16. Cette situation est caractérisée par l'envoi des deux ordres *S3* et *S12* simultanément ce qui peut conduire à des blocages en exécution réelle.

Ce blocage est lié au non respect de la contrainte de sécurité ❼ qui interdit la sortie du vérin P3 tant que le vérin P1 n'est complètement pas rentré. La synthèse va permettre d'enlever ces situations incohérentes.

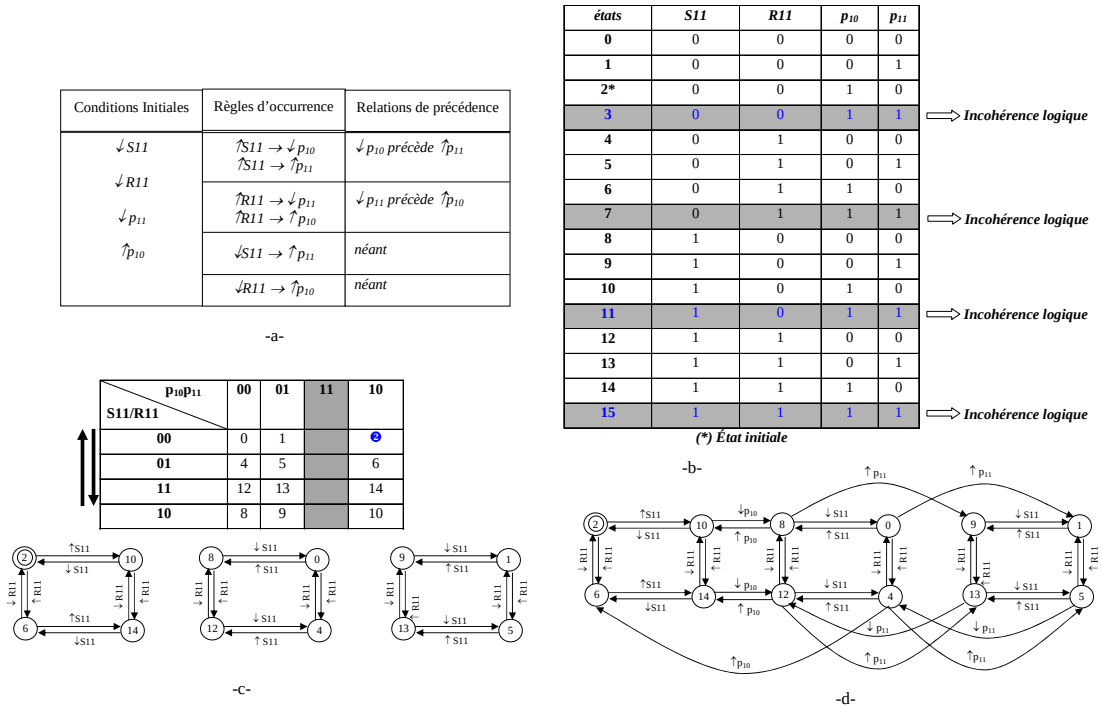
2.4 Modélisation structurée de la partie opérative.

Pour le système de tri de caisses, la décomposition modulaire de la partie opérative entraîne la distinction de cinq mouvements : le mouvement du poussoir P1, le mouvement du poussoir P2, le mouvement du poussoir P3 et celui le mouvement du poussoir P4.

Pour le mouvement du poussoir P1 (figure 5.5), deux événements commandables *S11* et *R11* sont définis, et p_{10} et p_{11} sont les deux événements non commandables. En situation initiale, le vérin se trouve en position entrée p_{10} et aucun ordre n'est envoyé à la partie opérative. Afin de définir les règles d'occurrence, il faut s'intéresser aux conséquences des ordres envoyés. L'activation de l'ordre *S11* permet au vérin de se déplacer vers la sortie et de quitter sa position entrée. De même, l'activation de *R11* fait revenir le vérin de la position sortie jusqu'à la fin de course p_{10}. Suite à l'activation de *S11*, la chronologie des événements non commandables est la suivante : désactivation de p_{10} puis activation de p_{11}. Pour la relation de précédence liée à l'activation de l'ordre *R11*, la chronologie se présente ainsi : désactivation de p_{10} avant activation de p_{11}. Les conditions initiales et les différentes relations concernant le mouvement horizontal sont résumées sur le tableau de la figure 5.5a.

Pour obtenir l'automate final du mouvement du poussoir P1, nous allons utiliser ici la méthode de construction booléenne qui consiste à obtenir à partir de la table de vérité, l'automate des évolutions commandables puis le complément de l'automate par ajout

des évolutions non commandables. Le mouvement est alors décrit de façon complète à travers 12 états et 36 transitions.



Conditions Initiales	Règles d'occurrence	Relations de précédence
$\downarrow S11$ $\downarrow R11$ $\downarrow p_{11}$ $\uparrow p_{10}$	$\uparrow S11 \rightarrow \downarrow p_{10}$ $\uparrow S11 \rightarrow \uparrow p_{11}$	$\downarrow p_{10}$ précède $\uparrow p_{11}$
	$\uparrow R11 \rightarrow \downarrow p_{11}$ $\uparrow R11 \rightarrow \uparrow p_{10}$	$\downarrow p_{11}$ précède $\uparrow p_{10}$
	$\downarrow S11 \rightarrow \uparrow p_{11}$	*néant*
	$\downarrow R11 \rightarrow \uparrow p_{10}$	*néant*

-a-

$p_{10}p_{11}$ / S11/R11	00	01	11	10
00	0	1		❷
01	4	5		6
11	12	13		14
10	8	9		10

-c-

états	*S11*	*R11*	p_{10}	p_{11}	
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 initiale

-b-

-d-

Figure 5.5. Construction de l'automate du mouvement du vérin P1.

Les automates des autres mouvements sont construits suivant la même méthode. La figure 5.6a illustre l'automate à états booléens du mouvement du poussoir P2. Les mouvements des poussoirs P3 et P4 sont illustrés dans les figures 5.6b et 5.6c

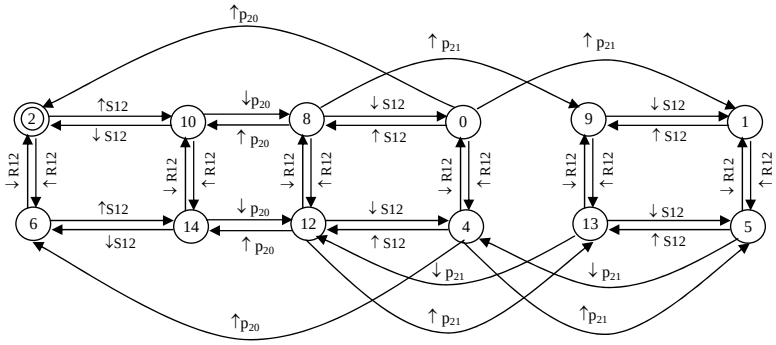


Figure 5.6a. Automate du poussoir P2.

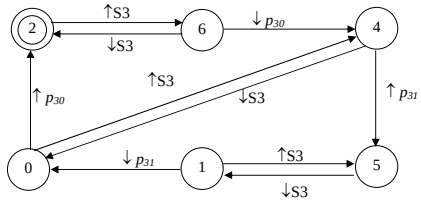


Figure 5.6b. Automate du poussoir P3.

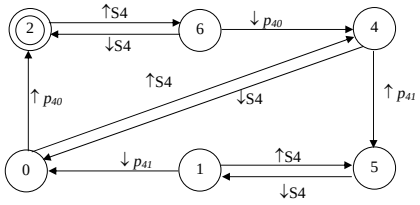


Figure 5.6c. Automate du poussoir P4.

La figure 5.7 présente un extrait du modèle final concernant le comportement de la partie opérative, après la composition asynchrone de ces différentes composantes. Celles-ci sont construites de la même manière. L'automate global comporte alors 15552 états et 163296 transitions.

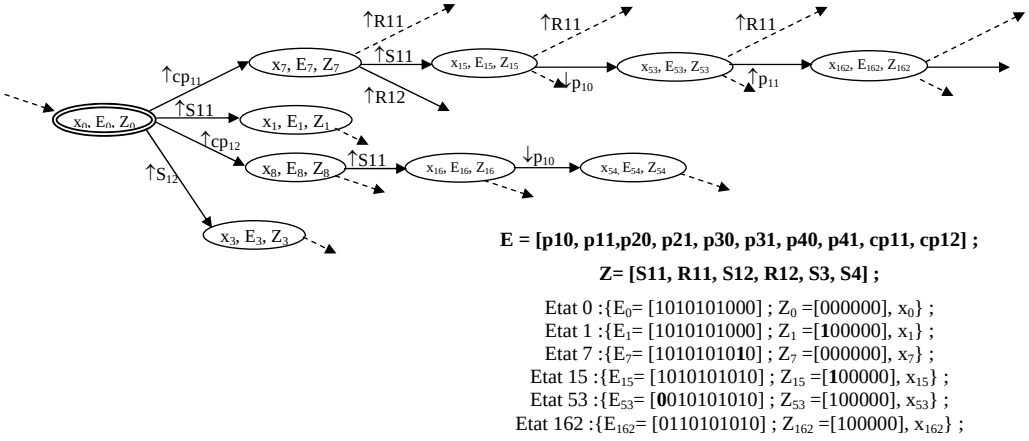


Figure.5.7. Automate à états booléens du procédé.

Le niveau de granularité souhaité et la méthodologie que nous utilisons n'étant pas restrictive et totalement indépendante des modèles de spécifications, elle génère un nombre d'états et de transitions relativement important au regard de la simplicité industrielle du système traité.

3 SYNTHESE DE LA COMMANDE

Une fois la phase de modélisation effectuée, on applique la démarche de synthèse préconisée pour obtenir une commande réactive, déterministe, la plus permissive vis-à-vis des contraintes et sans blocage.

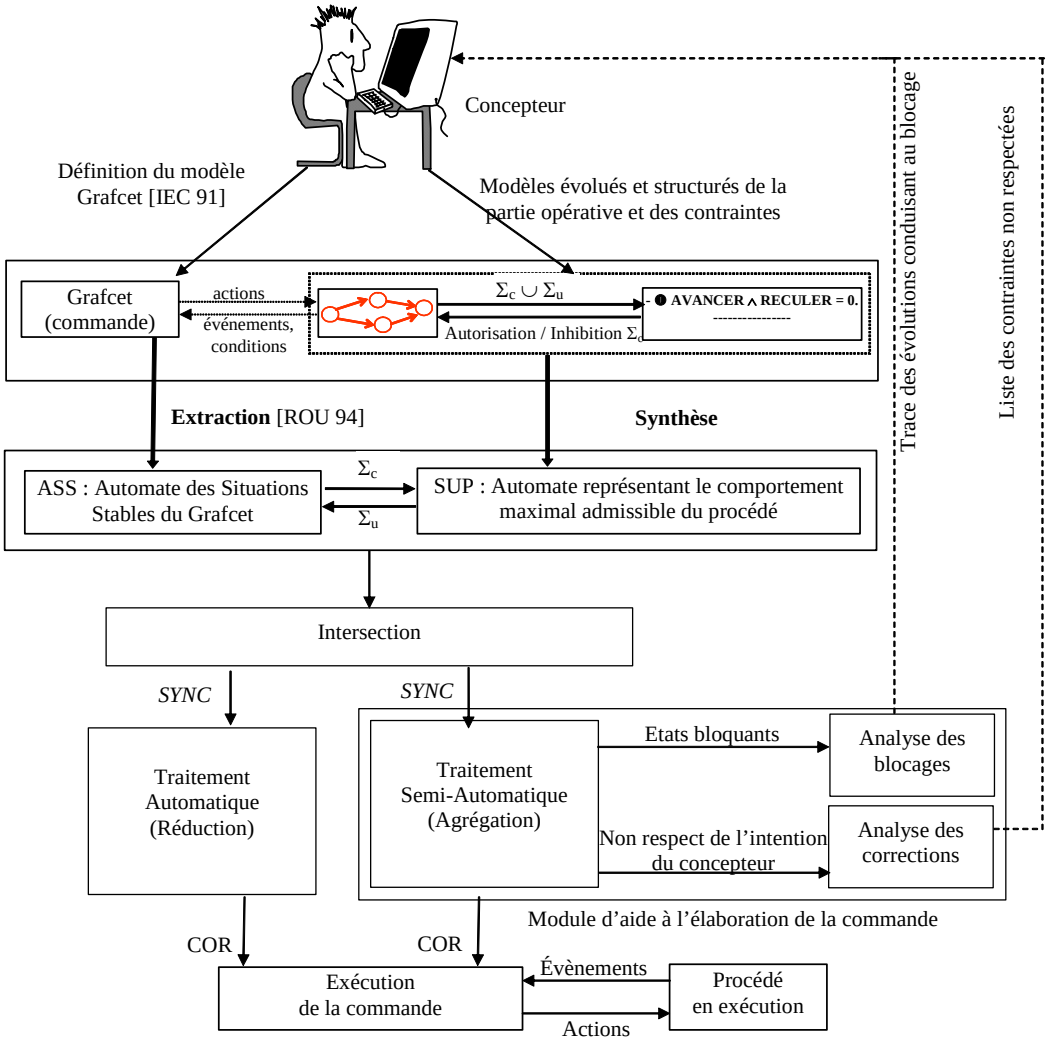


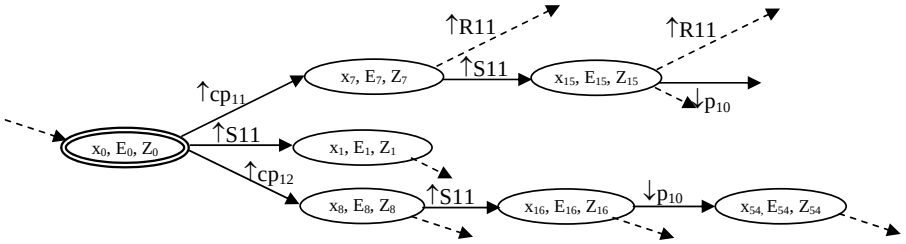
Figure 5.8. Démarche de synthèse de commande.

3.1 Synthèse de superviseur

L'algorithme de synthèse permet d'obtenir un automate de superviseur composé de 8747 états et de 76302 transitions, ses premiers états sont présentés par la figure 5.9

Cet automate décrit la façon dont les différentes évolutions de la partie opérative (figure 5.9) sont fixées afin de respecter les contraintes spécifiées. Par exemple, l'ordre S11 ne peut être autorisé si S3 est envoyé. Par conséquent, les propriétés sont satisfaites

concernant les premières évolutions de l'automate. En fait, l'algorithme de synthèse du superviseur garantit, pour l'ensemble des propriétés, que ces dernières sont respectées pour toutes les évolutions de l'automate.



E = [p10, p11,p20, p21, p30, p31, p40, p41, cp11, cp12] ; Z= [S11, R11, S12, R12, S3, S4] ;

Etat 0 :{E_0= [1010101000] ; Z_0 = [000000], x_0} ;Etat 1 :{E_1= [1010101000] ; Z_1= [**1**00000], x_1} ;
Etat 7 :{E_7= [101010101**0**] ; Z_7=[000000],x_7} ;Etat 15 :{E_{15}= [1010101010] ; Z_{15}= [**1**00000], x_{15}};
Etat 53 :{E_{53}= [**0**010101010] ; Z_{53} = [100000], x_{53}};…

Figure 5.9. Extrait de l'automate superviseur.

3.1.1 Génération de la commande

Même avec la structuration de l'étape de modélisation et l'adaptation de la démarche de synthèse, l'absence des situations bloquantes n'est pas garantie. Dans le cadre de cette application, l'étape d'intersection révèle l'existence d'une situation bloquante. Dans cette partie, nous appliquons la méthode de synthèse sous contrôle du concepteur en visualisant la séquence de blocage pour détecter, par analyse, l'origine du blocage par rapport aux modèles de départ. La séquence de blocage est présentée dans la figure 5.10a.

3.1.2 Synthèse sous contrôle du concepteur

- **Analyse et détection de l'origine du blocage**

La figure 5.11 présente les traces des évolutions de ce blocage pour deux pas de visualisation. Le traitement des blocages s'effectue soit en agissant sur le Grafcet, soit sur les contraintes et/ou la partie opérative. Ainsi, la figure 5.10a présente un état bloquant se situant dans la région r_{18}.

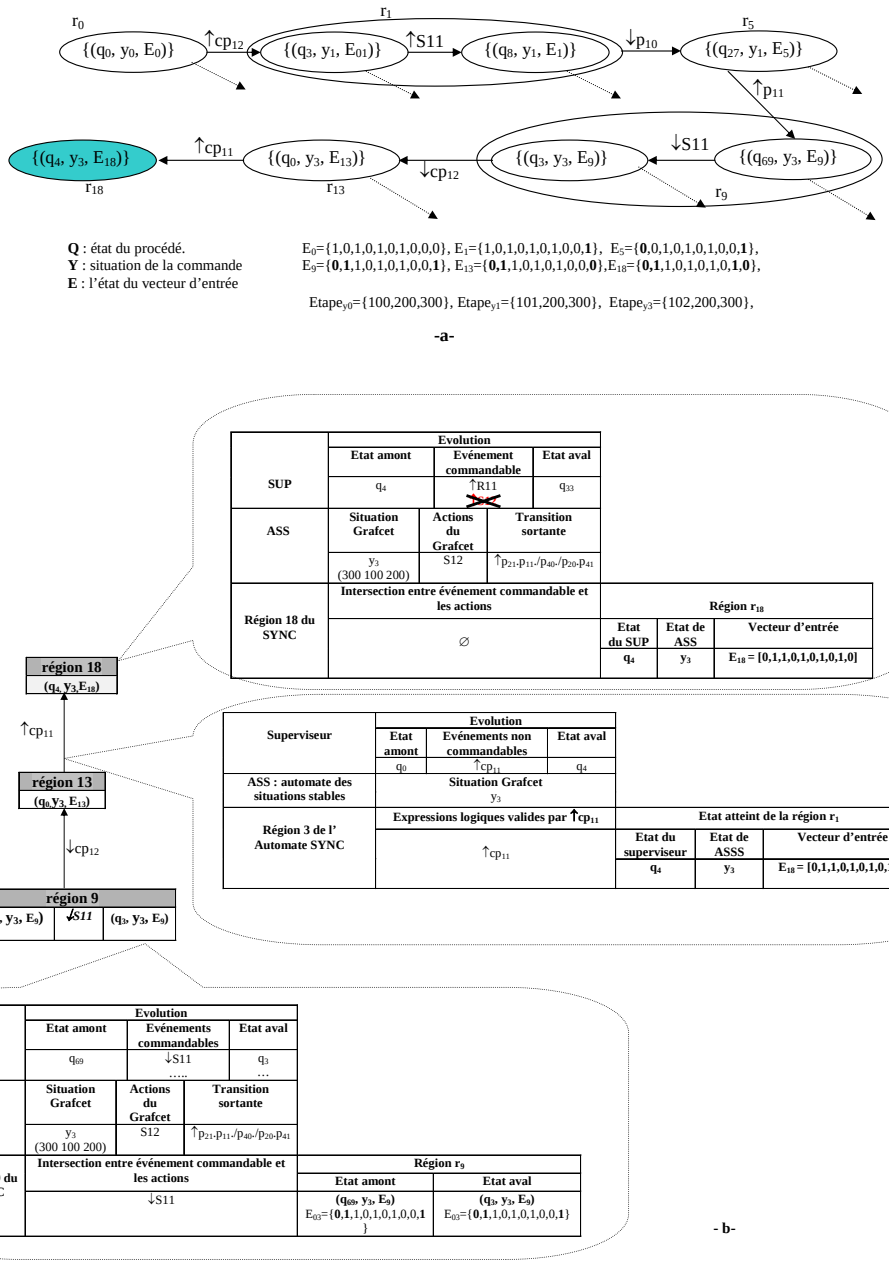


Figure 5.10 : a) Extrait de l'automate SEQ visualisé pas à pas depuis le blocage,
b) Analyse de la séquence bloquante.

Dans la situation bloquante (q_4, y_3, E_{18}), la commande se trouve en X102, X200 et X300 du Grafcet où l'ordre *S12* est envoyé vers la PO) (figure 5.10b). Pour quitter cette situation, il faut franchir la transition $p_{21}.p_{11}$ (Grafcets de spécifications figure 5.3). L'état du superviseur q_4 est caractérisé par l'interdiction de l'activation de *S12*, ce qui conduit à une intersection vide entre les ordres actifs de ASS et les événements commandables autorisés par le superviseur. En remontant la séquence du blocage, le pseudo état qui permet d'atteindre l'état bloquant est l'état (q_0, y_3, E_{13}). L'évolution de

celui-ci à l'état bloquant est définie par l'événement $\uparrow cp_{11}$, cela signifie que le système détecte l'arrivée d'une petite caisse. Or, la contrainte ❺ interdit la sortie du vérin P2 en présence d'une petite caisse. Nous pouvons dire alors que cette contrainte est restrictive par rapport au bon fonctionnement du système.

La solution que nous proposons est de relancer la démarche de synthèse hors ligne sans prendre en compte cette contrainte. Nous constatons que la commande générée est exempte du blocage (figure 5.11). Nous obtenons un automate de commande de 560 états et 2093 transitions. Quelques pas d'évolutions de cet automate sont présentés dans la figure 5.11.

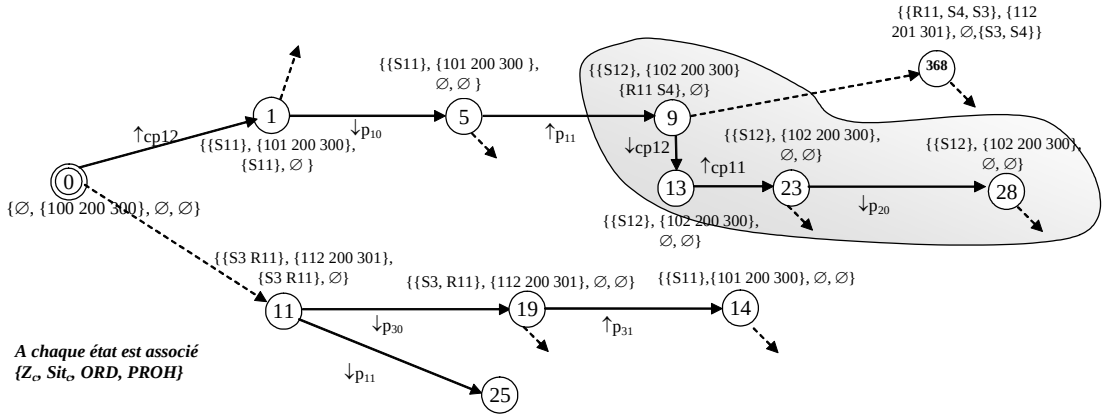


Figure 5.11 : Extrait de l'automate de commande OPT après correction.

3.2 Influence de la modélisation booléenne

Pour démontrer l'intérêt de la modélisation booléenne, nous avons repris la démarche initiale avec les contraintes ❸, ❹ et❼. En terme d'explosion combinatoire et en temps de calcul, la méthode de synthèse, selon Kumar [KUM 91], basée sur l'utilisation des automates à états rudimentaire pour la PO et les contraintes, a permis d'obtenir un automate superviseur de 16524 états et de 153432 transitions. Par contre, la 2ème méthode basée sur l'utilisation des modèles évolués, automates booléens et équations logiques pour les contraintes, a donné un superviseur avec un nombre d'états et de transitions inférieurs, de façon très significative par rapport à la 1ère. Le tableau 5.1 montre le gain en nombre d'états obtenus.

	automates à états (KUMAR)	**Automates booléens +Equation logique**	**Gain**
Synthèse du superviseur	16524 états	8747 états	53 %
	153432 transitions	76302 transitions	50 %

Tableau 5.1. Influence de la modélisation Booléenne sur la synthèse.

4 CONCLUSION

Dans ce chapitre, nous avons illustré à travers un exemple réel d'une complexité moyenne, comment prendre en compte des modèles évolués et structurés (des automates à états booléens construits à base de règles pour la PO et des équations logiques pour les contraintes) dans un objectif de synthèse de commande optimale, implantable sur Automate Programmable Industriel (API). En plus, nous avons montré que la démarche de synthèse, étant particulièrement sensible à l'intuitivité et aux erreurs de modélisation, un principe de traitement des blocages, non plus automatique, mais reposant sur l'alternance entre la synthèse et l'analyse des blocages, a été appliqué pour résoudre ce problème.

Cette application a mis en évidence l'intérêt d'utiliser des modèles évolués pour la PO et les contraintes en terme d'explosion combinatoire, en bénéficiant d'une réduction très significative en nombre d'états.

L'intégration du concepteur dans la boucle de synthèse permet donc de mettre en évidence l'intérêt d'effectuer une analyse de la commande générée pour éviter toute contradiction entre les objectifs désirés par la commande et les modèles utilisés.

CONCLUSION GENERALE.

Les travaux présentés dans ce mémoire portent sur la synthèse de commande des systèmes à événements discrets (SED) et son implantation correcte à partir d'une spécification Grafcet, d'un modèle de partie opérative et d'un modèle de contraintes.

Cette démarche originale utilise le Grafcet sans restrictions particulières. Il s'agit de générer la commande correspondante à la restriction minimale du comportement du Grafcet qui garantit la conformité par rapport aux contraintes imposées. La prise en compte d'une modélisation de la partie opérative assure un comportement correct du Grafcet lors de l'exécution réelle.

La description précise du comportement de la partie opérative est une opération complexe et difficile à concevoir. Pour contourner les difficultés d'une telle modélisation, nous avons choisi une démarche modulaire permettant d'exprimer des causalités simples entre les éléments de la partie opérative et nous avons proposé une méthode structurée à base de règles, pour aider le concepteur dans sa tâche de conception des modèles de partie opérative. Pour les contraintes, nous avons adopté une modélisation par équations logiques dans l'algèbre de Boole classique en termes d'entrées/sorties de la commande.

Deux variantes de la démarche de synthèse sont proposées : synthèse de commande contrainte et synthèse de commande supervisée. La synthèse de commande contrainte consiste à mettre en œuvre une démarche intuitive et progressive permettant de passer de la spécification à l'implantation de la commande. Cette mise en œuvre a montré l'intérêt d'une modélisation par équations logiques et l'amélioration apportée à la démarche classique. Cependant, elle reste assez limitée. En effet, en plus de la difficulté de sa conception, cette méthode est confrontée aux problèmes liés à la prise en compte d'un certain type de contraintes se rapportant seulement aux ordres actifs dans une situation du Grafcet, sans celles

liées aux ordres non actifs. La deuxième variante consiste à obtenir une commande réactive, déterministe, sans blocage et sûre vis-à-vis des contraintes de sûreté et de vivacité liée cette fois aux ordres actifs et non actifs en utilisant les acquis de la théorie de supervision. Cette méthode a aussi permis de conserver les apports de la nouvelle modélisation dont la réduction en explosion combinatoire obtenue par le développement d'un algorithme de synthèse du superviseur adapté à la modélisation évoluée est adoptée.

L'ensemble de ces travaux de thèse a permis d'identifier plusieurs perspectives qui, associées aux problèmes dégagés au cours de notre étude, constituent des pistes intéressantes et prometteuses pour des travaux de recherche à venir. Ces pistes concernent :

- L'extension de la 1ère variante proposée dans ce mémoire afin de dépasser ces limitations et prendre en compte toutes les contraintes de sûreté et de vivacité. Il s'agit d'étendre l'étape d'extraction de ASS en mémorisant dans chaque état de ASS, en plus des actions actives, les actions non actives ou désactivées. Ceci nécessitera d'étendre les opérations d'intersection pour obtenir une commande sûre, déterministe, réactive et sans blocage.
- Le développement d'un module de diagnostic interprétant les éléments en provenance du procédé. En cas d'occurrence d'événements non prévus, ce module permettrait de relancer la synthèse de manière à générer une nouvelle commande correspondante à un fonctionnement en mode dégradé.
- L'intégration du temps dans la démarche en développant une méthode plus adaptée. Cette intégration va conduire à utiliser d'autres modèles pour la partie opérative et les contraintes (tels les automates temporisés ou les RdP temporisés). Ceci nécessite de développer une méthode de synthèse à travers la proposition d'une intersection globale plus adaptée aux nouveaux modèles.
- La mise au point d'un module simulateur de modèles (partie opérative, contraintes et Grafcet) permettant l'interprétation à un niveau d'abstraction plus élevé des séquences bloquantes visualisées en vue de faciliter à l'utilisateur l'analyse pour détecter l'origine de blocage.
- Le développement d'une bibliothèque de modèles éprouvés, liée aux différents éléments de la partie opérative conformément à la sémantique préconisée par notre approche et au degré de la granularité des systèmes traités.

Les travaux présentés et les perspectives montrent la nécessité d'avoir un concepteur très aguerri dans la phase de modélisation. Par conséquent, la démarche de synthèse semi-automatique telle que nous l'avons présentée au chapitre 4 pourrait être détournée de son objectif initial, à savoir la construction d'une commande correcte au profit d'une opération de validation de la commande. Dans ce cas, le concepteur ne serait plus unique : la conception des modèles des contraintes et de la partie opérative serait à la charge d'un expert qui possède la connaissance du système, la conception de la commande sous forme de Grafcet étant à la charge du concepteur.

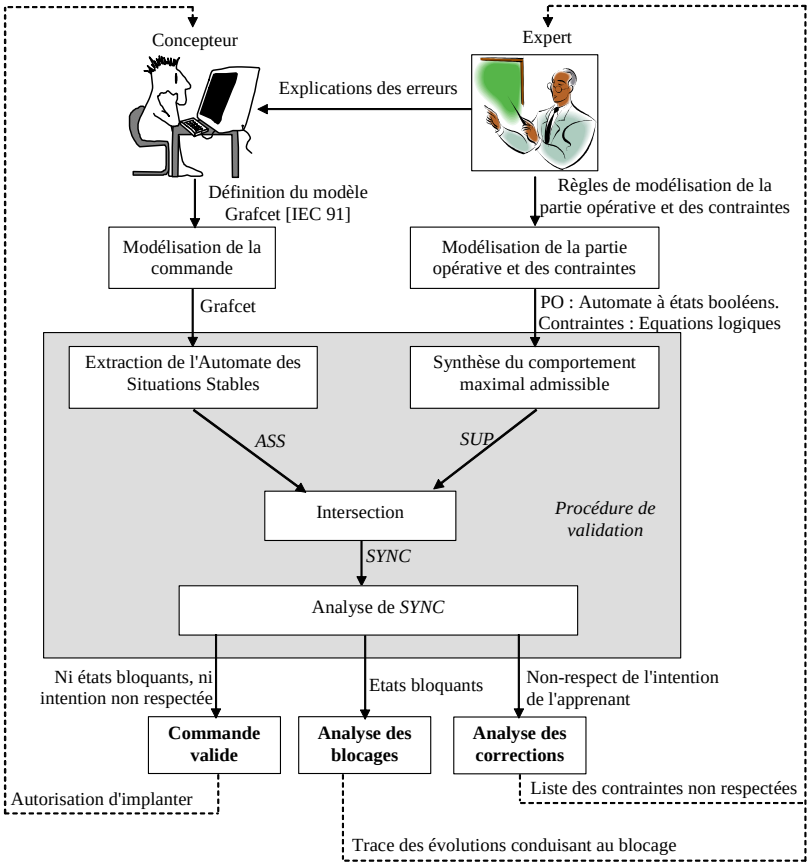


Figure 6.1 : Principe de la validation de commande

La figure 6.1 illustre ces propos. En tenant compte du cahier des charges, le concepteur élabore sa commande et la transmet à la procédure de validation, l'expert ayant défini au préalable les différents modèles. Deux cas peuvent alors se produire à l'issue de l'opération d'intersection :

- La commande respecte les spécifications imposées au système, alors le concepteur est autorisé à implanter son Grafcet dans un A.P.I. en toute sécurité, l'expert est ainsi certain qu'il n'y a aucun risque pour le matériel opératif, les opérateurs ou le produit.

- La commande délivrée par le concepteur n'est pas correcte et la procédure propose alors à l’expert une alternative. Il s’agit :
    - D’une situation de blocage. L’expert analyse la situation bloquante grâce aux informations élaborées à partir de la simulation des différents modèles. L’expert renseigne ensuite le concepteur quant aux raisons du blocage, lui permettant ainsi d’effectuer des corrections sur le grafcet de commande.
    - Du non-respect des intentions du concepteur. En effet, il existe des corrections générées pour prendre en compte les contraintes imposées au système. L’expert appréhende une liste de contraintes non respectées et en déduit une explication pour le concepteur lui permettant de revoir son grafcet de commande.

Une nouvelle itération de la procédure de validation sera ensuite effectuée à partir du modèle corrigé.

Cette procédure pourrait facilement s’orienter vers le domaine de l’enseignement des automatismes industriels en remplaçant l’expert par l’enseignant et le concepteur par l’apprenant.

BIBLIOGRAPHIE

[AFN 82] Afnor, Diagramme fonctionnel Grafcet pour la description des systèmes logiques de commande, Norme française NF C03-190, Paris, NFC 1982 et 1990.

[BAL 93]. S. Balemi, G. J. Hoffman., P. Gyugyi, H. Wong-Toi, G.F. Franklin, « Supervisory control of a rapid thermal multiprocessor », IEEE Transactions on Automatic Control, vol. 38, n°7, 1993, pp. 1040-1059.

[BER 99] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen, Systemsand Software Verification: Model-Checking techniques and tools, Heidelberg, Springer-Verlag Editions, 1999, ISBN : 3-540-41523-8

[BLA 79] M. Blanchard, Comprendre, maîtriser et appliquer le GRAFCET, Paris, Editions CEPADUES, 1979.

[BOU 02] E. Bourdon, M. Lawford, W.M. Wonham. “Robust Nonblocking Supervisory Control of Discrete-Event Systems”, Proceeding of American Control Conference, ACC’ 02, pp 730-735, 2002.

[BRA 00] B. A. Brandin, R. Malik, P Dietrich, Incremental system verification and synthesis of minimally restrictive behaviours. American Control Conference, ACC’00, Chicago, USA, pp. 4056-4061., 2000.

[BRA 96] B. A Brandin,.. The real-time supervisory control of an experimental manufacturing cell, IEEE Transactions on Robotics and Automation, 12(1), pp 1–14, 1996.

[CAR 00] V Carre-Menetrier, J. Zaytoon, Démarche progressive d’implantation d’une commande sûre dans un API. A partir d’une spécification Grafcet. Revue de l’électricité et de l’Electronique R.E.E., N°3, mars 2000, pp 69-79.

[CAR 01] V Carre-Menetrier, J. Zaytoon, **A .Tajer**, Formal composition algorithm for the synthesis of a correct controller implementation", IFAC INCOM 2001 Symposium, Vienne-Autriche, 20-22 septembre 2001.

[CAR 02] V Carre-Menetrier, J. Zaytoon, Grafcet : behavioral issues and control synthesis, European Journal of Control, vol. 8, n° 4, 2002, p. 375-401.

[CAS 99] C. G. Cassandras, S. Lafortune: Introduction to Discrete Event Systems, Kluwer Academic Publishers, 1999.

[CHA 00a] S. Chafik, E. Niel, Du contrôle par supervision hièrarchique et distribuée: application à la coordination, thèse de Doctorat de l'institut National des Sciences Appliquées de Lyon, 2000.

[CHA 00b] S. Chafik, E. Niel, Hierarchical-decentralized solutions of supervisory control. 3th International Symposium on Mathematical Modelling, MATHMOD, Vienna, Austria. 2000,

[CHA 01] V. Chandra. and R. Kumar, A Discrete Event Systems Modeling Formalism based on Event Occurrence Rules and Precedences . IEEE Transactions on Robotics and Automation , 2001,

[CHA 02] V. Chandra and R. Kumar, ``A Event Occurrence Rules based Compact Modeling Formalism for a Class of Discrete Event Systems" . Mathematical and Computer Modeling of Dynamical Systems , pp 49-73, volume 8, n° 1, 2002.

[CHA 96] F Charbonnier," Commande supervisée des systèmes à événements discrets", Thèse de doctorat de l'institut National Polytechnique de Lorraine, 1996.

[CHU 92] S.L Chung., S. Lafortune, F.Lin, Limited lookahead policies in supervisory control of discrete event systems. IEEE Transactions on Automatic Control, vol. 32, Issue : 12, p.1921-1935, December 1992

[FAB 94] M. Fabian, B. Lennartson. Petri Nets and Control Synthesis; An Object Oriented Approach. Proc of the 2nd IFAC/IFIP/IFORS Workshop on Intelligent Manufacturing Systems, IMS '94, Vienna, Austria, June 1994.

[FAB 98] M. Fabian, A. Hellgren. PLC-based Implementation of supervisory Control for Discrete Event Systems as Sequential Function Charts. Proc of CDC 98, Tampa, Florida, USA, December 1998.

[FEN 90] L. Fen, W. M. Wonham, 1990, Decentralized control and coordination of discrete-event systems with partial observation, IEEE Transactions on Automatic Control, 35, 12.

[FUS 83] A.Fusaoka, H. Seki, K. Takahashi., A description and reasoning of plant controllers intemporal logic. Proceedings of the 8th International Joint Conference on Artificial Intelligence,Karlsruhe, Germany, 1983

[GAF 96] D. Gaffé, le modèle grafcet : réflexion et intégration dans une plate forme multiformalisme synchrone, PhD Thesis, Université de Nice-Sophia Antipolis, Janvier 1996.

[GAF 03] D. Gaffé, « Modélisation et vérification d'un système mécatronique par SyncCharts » Actes de MSR 03, pp 95-107, 6-8 octobre 2003, Metz

[GHA01]. A. Ghaffari., N. Rezg, X. Xie, Conception du superviseur optimal vivant à l'aide de la théorie des régions, Modélisation des Systèmes Réactifs MSR'2001, Toulouse, 17-19 octobre 2001

[GHA02]. A. Ghaffari, N. Rezg, X. Xie, Algebraic and geometric characterization of Petri net controllers using the theory of regions, Proceedings of 6th International Workshop on Discrete Event Systems, WODES'02, Saragoza, Espagne, 2-4 octobre 2002

[GIU 96] A. Giua, Petri net techniques for supervisory control of discrete event systems. In : Proceedings of 1st Int. Workshop on Manufacturing and Petri nets, pp. 1-30.- Osaka, Japan, 1996.

[GOD 96] A. Godon., Contribution à la commande des systèmes à événements discrets par réseaux de pétri, Thèse de doctorat de l'université d'Angers,1996.

[GOH 98] P. Gohari., W.M. Wonham, 1998, A linguistic framework for controlled hierarchical DES. Proceedings of the 4th IEEE Workshop On Discrete Event Systems, WODES 98,Calgary.

[GOU02]. D. Gouyon, J.F. Petin, A. Gouin, « Modèles du procédé et de ses spécifications pour la synthèse de la commande» Actes de MSR 03, pp 45-60, 6-8 octobre 2003, Metz (France).

[GOU 03] D. Gouyon., J.F Petin., A. Gouin, « Modèles du procédé et de ses spécifications pour la synthèse de la commande» Actes de MSR 03, pp 45-60, 6-8 octobre 2003, Metz.

[GOU 04] D. Gouyon., Contrôle par le produit des systèmes d'exécution de la production : apport des techniques de synthèse, Thèse de l'université Henri Poincaré, soutenue le 6 décembre 2004, Nancy-I.

[HAN 97] H.-M. Hanisch, J. Thieme, A. Luder, Towards a synthesis method for distributed safety controllers based on net condition/event systems, Journal of Intelligent Manufacturing, 8, pp. 357-368, 1997

[HEL 01] A. Hellgren, M. Fabian, B. Lennartson. Modular Implementation of Discrete Event Systems as Sequential Function Charts Applied to an Assembly Cell. Proceedings of the 2001 IEEE Conference on Control Applications, Mexico City, Mexico, September 2001.

[HEL 02] A Hellgren., B. Lennartson, M. Fabian, Modelling and PLC-based implementation of modular supervisory control, Proceedings of 6th International Workshop on Discrete Event Systems, WODES'02, Saragoza, Espagne, 2-4 octobre 2002

[HOL 90] L E Holloway, B H. Krogh, Synthesis of feedback logic for a class of controlled Petri nets [J]. IEEE Trans on Automat Control, 1990, 35(5): 514-523.

[HOL 97] L.E. Holloway, B.H. Krogh, A. Giua, A survey of Petri net methods for controlled discrete event systems. Discrete Event Dynamic Systems: Theory and Applications, vol. 7, 1977, pp. 151-190.

[HOP 79]. J. Hopcroft, J. Ullman, Introduction to automata theory, languages, and computation, Addison-Wesley, 1979.

[IEC 88] International Electrotechnical Commission, Preparation of function charts for control systems », International Standard, CEI/IEC 848, 1988.

[IEC 91] International Electrotechnical Commission, « Preparation of function charts for control systems », International Standard, CEI/IEC 848, 1991 (revised version).

[IEC 93] International Electrotechnical Commission, « PLCs – Part 3: programming

languages », Publication 611131-3, 1993.

[IEC 02] International Electrotechnical Commission, Grafcet specification language for sequential function charts,2002, IEC 60848.

[JAL 94] M. A. Jafari and T. O. Boucher, (1994), "A Rule Based System for Generating Ladder Logic Control Program From a High Level System Model," Journal of Intelligent Manufacturing, vol. 5, pp. 103-120.

[JIA 00] S. Jiang and R. Kumar, Decentralized Control of Discrete Event Systems with Specializations to Local Control and Concurrent Systems. IEEE Transactions on Systems, Man and Cybernetics, Part B, pp 653-660, volume 30, number 5, October 2000.

[KAT 04] B. Kattan. Synthèse structurelle d'un contrôleur basée sur le Grafcet. Thèse de doctorat soutenue au laboratoire d'automatique de grenoble. 3, septembre 2004

[KOU 96] T. Kouthon, J.D. Decotignie., S. Koppenhoefer, On distribution of Grafcet software, Proceedings of the IMACS/IEEE Symposium on Discrete Events and Manufacturing Systems CESA'96, Lille, pp. 498-506.

[KUM 91] R. Kumar., Supervisory Synthesis Techniques for Discrete Event Dynamical Systems, Thesis for Ph. D. Degree, Université du Texas, Août 91.

[LAU96]. S.C . Lauzon, A.K. Ma, J.K. Mills , B. Benhabib, " Application of discrete event system theory to flexible manufacturing", IEEE control systems, p. 41-47, February 1996.

[LAU97]. S.C . Lauzon, A.K. Ma, J.K. Mills , B. Benhabib, An Implementation Methodology for the Supervisory Control of Flexible Manufacturing Wokcells", Journal of Manufacturing Systems, vol. 16, n°2, p. 91-101, 1997.

[LES 98] J.J. Lesage, J. M. Roussel, J.M. Faure, P. Lhoste, J. Zaytoon, Réactivité et déterminisme du comportement temporel du grafcet, ADPM'98, 3ème Conférence Internationale sur l'Automatisation des Processus Mixtes, 1998, Reims, pp. 99-106.

[LHO 97] P. Lhoste, J. M. Faure, J. J. Lesage., J. Zaytoon, Comportement temporel du Grafcet, J.E.S.A., 31, n°4, 1997, Hermès, pp. 695-711.

[LI 94] Y. Li; W. M. Wonham, .Control of vector discrete-event systems. II. Controller synthesis;Automatic Control, IEEE Transactions on , Volume: 39 , Issue: 3 , March 1994 Pages:512 – 531

[LIN 88] F. Lin and W. M. Wonham. Decentralized supervisory control of discrete event systems. Information Sciences, 44:199-224, 1988.

[LIN 90] F. Lin and W.M. Wonham. Decentralized control and coordination of discrete-event systems with partial observation. IEEE Transactions of Automatic Control, 35(12):1330--1337, December 1990.

[MAR 98] M.T Marikar., G.E Rotsein., S. Macchietto., 1998, An integrated environment for the design of procedural controllers. Volume II of 9th IFAC/INCOM symposium, Nancy.

[MOO 94] J.O..Mooly, K. Yamalidou, M.D. Lemmon, P. J. Antsaklis, Feedback control of Petri nets based on place invaraints. In: Proceedings of the 33rd IEEE Conference on Decisison and Control, pp.3104-3109.- Lake Buena Vista, FL, USA, December 1994.

[NDJ 99] C. H. Ndjab, synthèse de la commande des SED par Grafect, thèse de doctorat de l'université de Reims Champagne Ardenne, soutenue en 1999.

[PAR 00] E. Park, E. D. Tibury, and P. P. Khargonekar, "Control logic generation for machining systems using Petri net formalism," in Proc. 2000 IEEE Int. Conf. Systems, Man, and Cybernetics, vol. 5, Nashville, TN, 2000, pp. 3201–3206.

[PHI 03a] A. Philippot**.,** **A. Tajer**, F. Gellot., V. Carré-Ménétrier, « Synthèse de la commande spécifiée en Grafcet : application à un préhenseur pneumatique » Actes de MSR 03, pp 61-75, 6-8 octobre 2003, Metz

[PHI 03b] A. Philippot, **A. Tajer**, F. Gellot, V. Carré-Ménétrier: On-line synthesis approach based on a structured plant modeling, WODES 2004, Reims (France)

[PHI 04a] A. Philippot, **A. Tajer**., F.Gellot, V.Carré-Ménétrier, « Méthodologie de modélisation dans le cadre de la synthèse formelle des SED ». CIFA 2004, Douz (Tunisie)

[PHI 04b] A. Philippot, **A. Tajer,** F. Gellot, V. Carré-Ménétrier **:** Approche de synthèse en ligne basée sur une modélisation structurée de la partie opérative : SEE « Analyse et commande » volume1 N°3 troisième trimestre 2004.

[PHI 05] A. Philippot, **A. Tajer**, F.Gellot, V.Carré-Ménétrier**:** Méthodologie de modélisation dans le cadre de la synthèse formelle des SED: SEE « SED » SEE à paraître 2005

[PEN 04] S. S. Peng, M. C. Zhou. Ladder Diagram and Petri Net based Discrete Event Control Design Methods. In IEEE Transaction on systems, Man and Cybernetics. 2004

[QUE 00] M. H. de Queiroz, J. E. R. Cury, Modular Supervisory Control of Large Scale Discrete Event Systems, Proceedings of the Fifth Workshop on Discrete Event Systems (WODES 2000), Ghent, Belgium, 21-23 August, 2000

[QUE 02] M. H. de Queiroz, J E. R. Cury. Synthesis and Implementation of Local Modular Supervisory Control for a Manufacturing Cell. Proc. 6th International Workshop on Discrete Event Systems (WODES'02)

[RAM 87] P.J. Ramadge, W.M. Wonham, Supervisory control of a class of discrete event processes, SIAM J. Contr. Optim., vol. 25, n°1, p. 206-230, 1987.

[RAM 89] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. Proc. IEEE, Special Issue on Discrete Event Dynamic Systems, 77(1):81--98, January 1989

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

[[ROU 94] J.M. Roussel, Analyse de Grafcets par génération logique de l'automate à états, Thèse de doctorat, Ecole Normale Supérieure de Cachan, 1994.

[ROU 96] J.M Roussel, J.J. Lesage, Validation and Verification of Grafcets using finite state machine, IMACS/IEEE CESA'96 Symposium on Discrete Events and Manufacturing Systems, 1996, Lille, pp. 765-770.

[SAN 95] A. Sanchez, S Macchieto., Design of procedural controllers for chemical processes, Computers Chem. Engng. 19, pp. S381-S386, 1995

[SAK 03] J. Sakarovitch, Eléments de la théorie des automates, Vuibert, ISBN 2-7117-4807-3, 2003.

[SHA 95] M. A. Shayman, R. Kumar. "A New Framework for Supervisory. Control of Discrete Event Systems: 1995. Technical Research Report 95-72, institute for systems Research, 1995

[TAK 02] S. Takai and T. Ushio, "A Modified Normality Condition for Decentralized Supervisory Control of Discrete Event Systems", Automatica, Vol.38, No.1, pp.185-189, 2002.

[THI 96] J.G. Thistle, « Supervisory control of discrete event systems » Math. Comput. Modeling, Vol. 23, p. 25-53, 1996.

[TAJ 03] **A. Tajer**, A. Philippot, F. Gellot., V. Carré-Ménétrier, Contribution à l'amélioration de la praticabilité des approches formelle de synthèse de commande, Actes des JDA'03, 2003, Valenciennes, pp 239-244.

[TAJ 04] **A. Tajer,** A. Philippot, F. Gellot, V. Carré-Ménétrier : Démarche Formelle de synthèse d'une commande sûre à partir d'une spécification Grafcet, CIFA 2004, 22-24 Novembre, Douz (Tunisie)

[THI 00] C. Thierry, J. M. Roussel, J.J. Lesage, A extended boolean algebra for the control of logical systems, 16th IMACS World Congress 2000, Lausanne, Switzerland - August 21-25, 2000

[TZA 02] S. G. Tzafestas, M. G. Pantelelis, and D. L. Kostis, "Design and implementation of logic controller using Petri nets and ladder logic diagrams,"SAMS, vol. 42, no. 1, pp. 135–167, 2002.

[UTE 93] Union Technique de l'Electricité (UTE), « Établissement des diagrammes fonctionnels pour systèmes de commande - Diagramme fonctionnel GRAFCET. Extension des concepts de base » Documentation de référence UTE C03-191, Juin 1993, 35 pages.

[WNG 98] K. C. Wong and W. M. Wonham. Modular Control and Coordination of Discrete-Event Systems. Discrete Event Dynamic Systems, 8(3):241-273, October 1998.

[WON 02] W.M. Wonham– Notes on control of discrete event systems. – juillet 2002. Systems Control Group, Dept. Of Electrical and computer Engineering, University of Toronto

[WON 87] W. M. Wonham, P.J. Ramadge, « On the supremal controllable sublanguage of a given language », SIAM J. Control Optimization, 25, 1987, p. 637-659.

[WON 88] W.M. Wonham, and P.J. Ramadge (1988). Modular supervisory control of DESs. Mathematics of control of discrete event systems, 1(1):13-30.

[WAN 00] Y. Wang, , "Supervisory Control of Boolean Discrete-Event Systems", M.A.Sc. Thesis, Dept. of Electl. & Cmptr. Engrg., Univ. of Toronto, June 2000.

[YAM 96] K . amalidou, J.O. Mooly, M.D. Lemmon, P. J. Antsaklis, Feedback control of Petri nets based on place invaraints. Automatica, vol. 32, n1, 1996.

[YOO 02] T.S Yoo., S. Lafortune, A general architecture for decentralized supervisory control of discrete-event systems, Discrete Event Dynamic Systems: Theory and Applications, 12, 2002.

[ZAY 97] J. Zaytoon, J. J. Lesage, L. Marce, J. M. Faure, P. Lhoste, Vérification et validation du Grafcet, J.E.S.A., 31, n°4, 1997, Hermès, pp. 713-740.

[ZAY 99] J. Zaytoon, V. Carre-Mnentrei., Grafcet et Graphe d'états: comportement, raffinement, vérification et validation. Journal Européen des systèmes automatisés, vol. 33, n°7, 1999, pp 751-782.

[ZAY 01] J. Zaytoon, A contribution to the validation of Grafcet controlled systems », European Journal of Control, vol. 7, 2001.

[ZHO 98] M.C. Zhou, M.D. Jeng, Modeling, analysis, simulation, scheduling, and control of semiconductor manufacturing systems - a Petri net approach.In: IEEE Trans. on Semiconductor Manufacturing, Vol. 11, No. 3, pages 333-357. 1998.

[ZHO 90] H. Zhong, W.M. Wonham, On the Consistency of Hierarchical Supervision in Discrete-Event Systems, IEEE Trans. on Aut. Control, 35(10):1125-1134. Hermes, 1989.

[ZHO 93] C. Zhou, F. DiCesare, Petri Net Synthesis for Discrete Event Control of Manufacturing Systems, Boston, Kluwer, MA, 1993.

ANNEXE 1

Algorithmes de Synthèse de Commande *COR*

1. INTERSECTION

L'automate SYNC est généré par l'algorithme itératif suivant, qui s'arrête lorsque aucune nouvelle région ne peut plus être créée :

Etape 1 : Développer la région initiale r_0 à partir de l'état st_0.

Etape 2 : Pour chaque région développée (par l'étape 1 ou l'étape 2-b) :

2-a : créer le premier état de chacune de ses régions aval,

2-b : développer chacune de ces régions aval.

Cet algorithme reçoit en entrée un automate $S = (\Sigma, Q, \Delta, q_0)$ correspondant au superviseur généré par l'étape de synthèse, un automate $ASS = (E, S, Y, T, y_0)$ équivalent du Grafcet et la structure initiale de l'automate SYNC donnée par $(\Sigma, \{st_0\}, \varnothing, st_0, \{st_0\}, \varnothing)$. Dans la suite, l'état courant de SYNC est caractérisé par l'état courant de S, la situation courante de ASS, ainsi que la configuration courante des valeurs des variables d'entrée.

L'algorithme itératif décrivant l'étape 1 (figure 1) consiste à créer les états de la région r_0 ainsi que les transitions les reliant. Ces transitions décrivent les événements commandables autorisés par le superviseur S et qui ne sont pas en contradiction avec les actions associées à la situation initiale du Grafcet. La première itération de la procédure crée les transitions commandables sortantes de l'état initial st_0 ainsi que leurs états aval qui sont ajoutés à la région initiale. Chacune des itérations suivantes continue le développement de la région r_0 à partir de l'un des états créés, lors des itérations précédentes, et ce jusqu'au traitement de tous les états de la région. Une itération est constituée de deux pas, l'un pour les ordres à activer et l'autre pour ceux à désactiver. Le premier pas consiste à tester les transitions commandables

sortantes de l'état courant "*q*" de S et correspondant à l'activation d'un ordre du Grafcet "$\uparrow$z". Lorsque cet ordre fait partie de ceux associés à la situation initiale du Grafcet, une transition correspondante à son activation est ajoutée à TR. En effet, dans ce cas, cette activation est prévue par la commande et est autorisée par le superviseur. L'état aval *st'* de la transition créée se distingue de l'état amont *st* uniquement par le premier élément du triplet *q'* qui correspond à l'état de S suite au franchissement de la transition correspondante. L'état st' est ensuite ajouté à la région r_0 s'il n'en fait pas déjà partie. Le deuxième pas de la procédure procède au traitement des événements commandables correspondants aux désactivations des ordres du Grafcet $\downarrow z$. Si l'ordre concerné *z* n'est pas inclus dans l'ensemble des ordres associés à l'état initial de ASS, alors la transition correspondante à $\downarrow z$ est créée dans r_0 pour indiquer que cet ordre est inhibé lors de la situation initiale de la commande.

$\forall st = (q, y_0, E_0) \in r_0$ Faire :
1- $\forall$ $(q, \uparrow z, q') \in \Delta$ tel que $(z \in Z_{y0})$ Faire :
- $TR = TR \cup \{ (st, \uparrow z, st') \}$; où $st' = (q', y_0, E_0)$
- Si $st' \notin r_0$ Alors $r_0 = r_0 \cup \{ st' \}$

2- $\forall$ $(q, \downarrow z, q') \in \Delta$ tel que $(z \notin Z_{y0})$ Faire :
- $TR = TR \cup \{ (st, \downarrow z, st') \}$; où $st' = (q', y_0, E_0)$
- Si $st' \notin r_0$ Alors $r_0 = r_0 \cup \{ st' \}$

Figure 1. Procédure de développement de la région initiale r_0

L'étape 2-b procède au développement d'une région r_c à partir de son premier état, généré lors de l'étape 2-a. L'algorithme correspondant (figure 2) est identique à celui utilisé pour définir la région initiale. Cependant, les états aval des transitions créées dans ce cas peuvent éventuellement appartenir à une région existante autre que la région étudiée. Dans ce cas, les deux régions sont fusionnées en r_c car elles correspondent à une même situation de la commande. L'opérateur $reg_{st'}$ retourne la région à laquelle appartient l'état st', et l'opérateur *éliminer(r)* effectue l'élimination de r de la partition des sous ensembles de ST.

$\forall st = (q, y, E) \in r_c$ Faire :
1- $\forall$ $(q, \uparrow z, q') \in \Delta$ tel que $(z \in Z_y)$ Faire :
- $TR = TR \cup \{ (st, \uparrow z, st')) \}$; où $st' = (q', y, E)$
- Si $(st' \in ST \wedge reg_{st'} \neq r_c)$ Alors $r_c = reg_c \cup r_{st'}$, éliminer($reg_{st'}$)
- Si $st' \notin r_c$ Alors $r_c = r_c \cup \{ st' \}$

2- $\forall$ $(q, \downarrow z, q') \in \Delta$ tel que $(z \notin Z_y)$ Faire :
- $TR = TR \cup \{ (st, \downarrow z, st') \}$; où $st' = (q', y, E)$
- Si $(st' \in ST \wedge reg_{st'} \neq r_c)$ Alors $r_c = reg_c \cup r_{st'}$, éliminer($reg_{st'}$)
- Si $st' \notin r_c$ Alors $r_c = r_c \cup \{ st' \}$

Figure 2. Procédure de développement d'une région r_c quelconque

L'étape 2-a est utilisée pour créer les transitions inter-régions qui représentent les événements non commandables sortant d'une région r_r récemment créée. Chaque itération de

la procédure correspondante (figure 3) est constituée de deux pas pour traiter les transitions non commandables sortantes de l'un des états *st* de la région r_r, et d'un troisième pas pour déterminer si cet état *st* est bloquant.

Le premier pas concerne le développement des transitions décrivant des événements non commandables correspondant aux fronts montants d'une variable d'entrée du Grafcet $\uparrow e$. Il s'agit des événements possibles dans l'état courant de S, autrement dit des réactions possibles du procédé. Chacun de ces événements est alors admis dans le comportement commun SYNC si la valeur courante de la variable *e* concernée vaut 0, c'est à dire si le front montant de cette variable peut physiquement avoir lieu. La transition correspondante créée dans SYNC mène à un état *st'*, où la valeur de *e* devient égale à 1. Cet état correspond à la situation de S suite à l'occurrence du même événement, tandis que la situation correspondante de ASS est mise à jour si l'expression logique *f(E)* associée à l'une de ses transitions aval devient vraie lors de l'occurrence de $\uparrow e$. Rappelons que cette expression porte sur les variables d'entrée du Grafcet et sur les fronts de ces entrées. Si l'état créé, *st'*, n'existe pas dans ST, alors il sera inclus au sein d'une nouvelle région.

Le deuxième pas de cette procédure est basé sur le même principe, il traite des transitions non commandables correspondant aux fronts descendants $\downarrow e$ des variables d'entrée du Grafcet. Ces transitions seront ajoutées à ST si la valeur courante de la variable *e* est à 1. Dans ce cas, elles mettent la valeur de cette variable à 0. Enfin, le troisième pas de la procédure consiste à ajouter l'état *st* faisant l'objet de l'itération en cours, dans la liste des états bloquants s'il n'a aucune transition sortante.

$\forall st = (q, y, E) \in r_r$ Faire :
1- $\forall (q, \uparrow e, q') \in \Delta$ tel que $(e \in E \wedge e=0)$ Faire :
Si $\exists (y, f(E), y') \in T)$ tel que f(E) =vrai
Alors
- TR=TR$\cup$\{ (st, $\uparrow$e, st') \} }; où st'=(q', y', E'), E' se distingue de E uniquement par la valeur de e qui devient 1 au lieu de 0
- Si st'$\notin$ST Alors r={st'}, ST=ST$\cup$r, $ETAPES_r=ETAPES_{y'}$
Sinon
- TR=TR$\cup$\{ (st, $\uparrow$e, st') \} }; où st'=(q', y, E'), E' se distingue de E uniquement par la valeur de e qui devient 1 au lieu de 0
- Si st'$\notin$ST Alors r={st'}, ST=ST$\cup$r, $ETAPES_r=ETAPES_y$
2- $\forall (q, \downarrow e, q') \in \Delta$ tel que $(e \in E \wedge e=1)$ Faire :
Si $\exists (y, f(E), y') \in T)$ tel que f(E) =vrai
Alors
- TR=TR$\cup$\{ (st, $\downarrow$e, st') \} }; où st'=(q', y', E'), E' se distingue de E uniquement par la valeur de e qui devient 0 au lieu de 1
- Si st'$\notin$ST Alors r={st'}, ST=ST$\cup$r, $ETAPES_r=ETAPES_{y'}$
Sinon
- TR=TR$\cup$\{ (st, $\downarrow$e, st') \} }; où st'=(q', y, E'), E' se distingue de E uniquement par la valeur de e qui devient 0 au lieu de 1
- Si st'$\notin$ST Alors r={st'}, ST=ST$\cup$r, $ETAPES_r=ETAPES_y$
3- Si $\neg(\exists$ (st, σ, st')$\in$TR) Alors BLOC = BLOC $\cup$\{ st \}

Figure 3. Procédure de création d'une nouvelle région

2. REDUCTION

L'objectif de cette étape est de générer l'automate COR qui représente la commande réactive et déterministe correspondant au plus grand sous-ensemble de comportements de SYNC, exempt de blocage. L'opération de réduction s'effectue en trois phases.

Dans la première phase, les états bloquants de SYNC sont rendus non atteignables par suppression de leurs transitions amont. Ces suppressions peuvent induire de nouveaux états bloquants qui seront alors traités de manière récursive. Les phases suivantes ont pour objectif de générer l'automate COR qui assure le déterminisme et la réactivité de la commande.

La deuxième phase dite d'optimisation consiste à retirer de l'automate SYNC - issu de la première phase - les transitions qui sont non atteignables durant l'exécution réelle, à cause de la nature réactive de la commande. En effet, la réactivité implique l'activation et la désactivation simultanée et instantanée de tous les ordres correspondant à l'ensemble des événements commandables d'une situation, et ceci avant toute occurrence d'une réaction du procédé. Ainsi, la deuxième phase procède au retrait des évolutions de SYNC représentant l'occurrence des événements non commandables à partir des états intermédiaires des régions de SYNC, c'est à dire les états dans lesquels l'ensemble des ordres correspondant à la situation courante de la commande n'est pas encore envoyé.

Pour assurer l'élimination effective de ces évolutions durant l'exécution, la troisième phase procède à l'agrégation des situations instables, de manière à aboutir à une commande réactive qui évolue, suite à l'occurrence d'un événement non commandable, d'une situation stable à une autre. Une situation stable correspond à une agrégation des états appartenant à une région de SYNC, suite au traitement du blocage ; elle contient les informations relatives aux ordres qui sont autorisés ou inhibés de manière instantanée, au sein de cette région.

2.1. *Traitement de la première phase*

Cette phase fait appel à la procédure *traiter-blocage()* (figure 4). Pour chaque état bloquant $st \in BLOC$, les transitions amont sont supprimées afin de rendre l'état non atteignable. Les incohérences, les blocages et les états non atteignables induits par ces suppressions, sont examinés ensuite et leur traitement dépend alors du type d'événement rencontré lors de la suppression des transitions ; il peut s'agir soit d'un événement non commandable ($\sigma \in \Sigma_u$), soit de l'envoi d'un ordre ($\sigma \in \uparrow Z$) ou soit de l'inhibition d'un ordre ($\sigma \in \downarrow Z$). A la suite de ces différentes opérations, l'état *st* est retiré de la liste des états ST et de la liste des états bloquants BLOC.

Les traitements relatifs à la suppression des différents types de transition sont les suivants :

- Si la transition supprimée désigne un événement non commandable, il faut rendre son état amont non atteignable car elle reflète une réaction du procédé qui ne peut être interdite par la commande. L'appel à la procédure *nettoyer(état)* permet d'effectuer ce traitement et de prendre en considération l'introduction d'éventuels blocages supplémentaires. Si, suite à la suppression de la transition, la région incluant l'état *st* est rendue non atteignable, c'est à dire si aucun de ses états ne possède plus de transition non commandable entrante, alors la procédure *retirer(r)* supprime cette région.

- Si la transition représente l'envoi d'un ordre $\uparrow Z$, alors cet ordre doit être interdit dans la situation courante de la commande. Par conséquent, il faut retirer de la région toutes les transitions représentant l'envoi de cet ordre. Si le retrait d'une de ces transitions rend son état aval non atteignable, la procédure *couper (état)* est alors invoquée pour éliminer cet état et traiter ses transitions aval. Enfin, si l'un des états amont des transitions retirées n'a pas d'autre transition sortante, il est ajouté à la liste BLOC.

- Si la transition correspond à l'inhibition d'un ordre $\downarrow Z$, cela implique que cet ordre n'est plus associé à la situation courante de la commande, le traitement consiste alors à retirer toute la région concernée. En effet, interdire uniquement la désactivation de l'ordre dans la région concernée aboutirait à la situation aberrante où un ordre est maintenu alors qu'il ne doit pas être exécuté. Avant de retirer la région, l'ensemble des transitions amont en provenance d'autres régions sont éliminées, et la procédure *nettoyer(état)* est invoquée pour traiter les états amont de ces transitions de manière à garantir la non atteignabilité de la région retirée.

∀ st∈BLOC Faire :
- ∀ t=(st', σ, st)∈TR Faire :
  - TR = TR-{ t }
  - Si $\sigma \in \Sigma_u$: Alors
    - nettoyer(st')
    - Si (∀ etat∈reg_{st} :¬ ∃ (State, τ, etat)∈TR tel que $\tau \in \Sigma_u$) Alors retirer(reg_{st})
  - Si $\sigma \in \uparrow Z$ Alors
    ∀ etat∈ reg_{st} Faire:
    - Si ∃ (etat, σ, etat') ∈TR Alors :
      - TR=TR-{ (etat, σ, etat') }
      - Si (¬ ∃ (st'', τ, état') ∈TR) Alors couper(état')
    - Si (¬ ∃ (etat, τ, etat'') ∈TR) Alors BLOC=BLOC ∪ {etat}
  - Si $\sigma \in \downarrow Z$ Alors
    - ∀(state, τ, état) tel que état∈reg_{st} Faire :
      - TR = TR-{ (state, τ, état) }
      - nettoyer(state)
    - retirer(reg_{st})
- ST=ST-{st} , BLOC=BLOC-{st}

Figure 4. Procédure traiter-blocage ()

L'algorithme décrit fait apparaître des procédures telles que *retirer(r)*, *couper(état)* et *nettoyer(état)* qui permettent de respectivement, retirer de l'automate SYNC une région r devenue non atteignable, retirer un état devenu non atteignable et traiter l'état amont d'une transition non commandable devant être retirée.

2.2. *Traitement de la deuxième phase*

La deuxième phase dite d'optimisation consiste à appliquer la procédure *optimiser ()* (figure 5) sur l'automate SYNC généré par la phase précédente. L'action de cette procédure consiste à retirer les transitions correspondant aux événements non commandables issus d'états ayant d'autres transitions aval représentant des événements commandables. Les régions et les états aval devenus non atteignables suite à ces retraits, sont supprimés.

∀ (st, α, st'')∈TR tel que $\alpha \in \Sigma_u$ Faire :
Si (∃ (st, σ, st')∈TR tel que $\sigma \in \Sigma_c$ Alors :
1- TR=TR-{ (st, α, st'') }
2- Si (∀ etat∈$reg_{st''}$: ¬ ∃ tr(état', τ, etat)∈TR tel que $\tau \in \Sigma_u$) Alors retirer($reg_{st''}$)
Sinon Si (¬ ∃ (state, τ, st'') ∈TR) Alors couper(st'')

Figure 5. Procédure optimiser ()

2.3. *Traitement de la troisième phase*

La troisième phase dite d'agrégation consiste à générer l'automate de commande COR dont les états correspondent aux régions de SYNC. Les évolutions intermédiaires représentant les états instables d'une région r_c sont remplacées par un état *c* décrivant la situation stable

correspondante de la commande. Cet état est muni de trois ensembles : les étapes de la situation correspondante du Grafcet, les ordres à autoriser et les ordres à inhiber. Les ordres à autoriser et à inhiber sont déduits des transitions commandables de la région agrégée, alors que les étapes actives du Grafcet sont déduites de l'état de S correspondant. La commande réactive ainsi générée est donnée par l'automate COR dont les états sont reliés par les transitions non commandables reliant les régions de SYNC.

Ainsi, à partir d'une situation, l'occurrence d'un événement en provenance du procédé fait évoluer l'automate immédiatement (sans avoir besoin de calculer des expressions logiques) vers une unique situation stable, et entraîne une autorisation/inhibition instantanée des ordres associés à cette situation. L'automate COR décrit donc un comportement déterministe et réactif correspondant au plus large comportement admissible de la commande, vis-à-vis des contraintes spécifiées. Il est défini par un 4-uplet (Σ_u, ET, ϕ, $etat_0$), avec:

- Σ_u l'ensemble des événements non commandables

- ET l'ensemble des états correspondants, chacun à une région de SYNC. A chaque état $et \in$ ET est associé le triplet (ACT_{et}, NAC_{et}, Sit_{et}) où :

  - ACT_{et} : représente l'ensemble des ordres à autoriser dans cet état,

  - NAC_{et} : représente l'ensemble des ordres à inhiber lorsque la commande se trouve dans cet état,

  - Sit_{et} : représente l'ensemble des étapes Grafcet correspondant à cet état.

- $\phi : \Sigma_u \times ET \rightarrow ET$ est une fonction partielle représentant les transitions de COR.

- $etat_0$ est l'état initial, dénommé 0.

La procédure qui effectue la phase d'agrégation (figure 6) reçoit en entrée l'automate SYNC réduit issu de la deuxième phase (Σ, ST, TR, st_0, r_0, $\varnothing$), et un automate COR initialisé à (Σ_u, ET=$\varnothing$, $\phi = \varnothing$, $etat_0$=0).

1- $\forall r_c \subset$ ST Faire :
  - ET=ET$\cup${c}
  - Sit_c=$ETAPES_{rc}$
  - $\forall$état$\in r_c$ Faire :
    - $\forall$(état,$\uparrow$z, état')$\in$TR Faire $ACT_c = ACT_c \cup$ z ,
    - $\forall$(état, $\downarrow$z, état'') $\in$TR Faire $NAC_c = NAC_c \cup$ z ,

-2- $\forall r_c, r_{c'} \subset ST^2$ Faire :
  - Si $\exists$(état,α, état')$\in$TR tel que état$\in r_c$ et état'$\in r_{c'}$ Alors : $\phi = \phi \cup \{(c, \alpha, c')\}$

Figure 6. Procédure agréger

ANNEXE 2

Algorithmes de synthèse en ligne : *SYNC*^{N}

Pour générer l'automate *SYNC*^{N}, l'étape d'intersection partielle est basée sur l'algorithme itératif qui s'arrête lorsqu'aucune fenêtre d'intersection ne peut être créée.

Etape 1 : Développer la fenêtre d'intersection partielle initiale f_0 à partir de la région initiale r_0 (figure 1).

Etape 2 : Développer une fenêtre d'intersection partielle quelconque f_c (figure 2).

Cet algorithme reçoit en entrée un automate SUP= (Σ, Q, Δ, q_0) correspondant au superviseur, un automate ASS = (E, S, Y, T, y_0) et la structure initiale de l'automate *SYNC*^{N} donnée par (Σ, {st_0}, $\varnothing$, st_0, {st_0}, {st_0}, $\varnothing$, $\varnothing$, $\varnothing$).

- **Développement de la fenêtre f_0**

a) ***Développement de la région initiale***

Le premier pas (❶) de l'algorithme itératif décrivant l'étape 1 (Figure. 1), consiste à créer les états de la région initiale r_0 ainsi que les transitions les reliant. La première itération de la procédure créée les transitions commandables sortantes de l'état initial st_0 ainsi que leurs états aval qui sont ajoutés à la région initiale. Chacune des itérations suivantes continue le développement de la région r_0 à partir de l'un des états créés lors des itérations précédentes, et ce jusqu'au traitement de tous les états de la région.

Une itération est constituée de deux pas, l'un pour les ordres à activer et l'autre pour ceux à désactiver. Le premier pas consiste à tester les transitions commandables sortantes de l'état courant « q » de S et correspondant à l'activation d'un ordre du Grafcet « ↑z ». Lorsque cet ordre fait partie de ceux associés à la situation initiale du Grafcet, une transition correspondante à son activation est ajoutée à TR. En effet, dans ce cas, cette activation est prévue par la commande et autorisée par le superviseur. L'état aval *st'* de la transition créée se

distingue de l'état amont *st* uniquement par le premier élément du triplet q' qui correspond à l'état de S suite au franchissement de la transition correspondante. L'état st' est ensuite ajouté à la région r_0 s'il n'en fait pas déjà partie.

Dans le deuxième pas de la procédure, on procède au traitement des événements commandables correspondants aux désactivations des ordres du Grafcet ↓z. Si l'ordre concerné z n'est pas inclus dans l'ensemble des ordres associés à l'état initial de ASS, alors la transition correspondante à ↓z est créée dans r_0 pour indiquer que cet ordre est inhibé lors de la situation initiale de la commande.

b) développement des transitions inter -région et d'une région quelconque

Dans le second pas (❷), tant que le nombre d'événements non commandables courant *Ncou* est inférieur ou égal au nombre d'événements non commandables *N*, il s'agit de :

A) Développer les transitions inter-régions

Les transitions inter-régions indiquent un changement de région, il faut donc créer le premier état de chacune de ces régions aval. Chaque itération de la procédure de création de ces transitions est constituée de deux pas pour traiter les transitions non commandables sortantes de l'un des états st de la région r_r et d'un troisième pas pour déterminer si cet état st est bloquant :

Le premier pas concerne le développement des transitions décrivant des événements non-commandables correspondants au front montant d'une variable d'entrée du Grafcet ↑*e*. Il s'agit des événements possibles dans l'état courant de S, autrement dit des réactions possibles du procédé. Chacun de ces événements est alors admis dans le comportement commun $SYNC^N$ si la valeur courante de la variable e concernée vaut 0, c'est-à-dire si le front montant de cette variable peut physiquement avoir lieu. La transition correspondante créée dans $SYNC^N$ mène à un état *st'*, où la valeur de *e* devient égale à 1. Cet état correspond à la situation de S suite à l'occurrence du même événement, tandis que la situation correspondante de ASS est mise à jour si l'expression logique *f(E)* associée à l'une de ses transitions aval devient vraie lors de l'occurrence de ↑*e*. Rappelons que cette expression porte sur les variables d'entrée du Grafcet et sur les fronts de ces entrées. Si l'état créé, *st'*, n'existe pas dans ST, alors il sera inclus au sein d'une nouvelle région.

Le deuxième pas de cette procédure est basé sur le même principe, il traite des transitions non commandables correspondant aux fronts descendants ↓*e* des variables d'entrée du

Grafcet. Ces transitions seront ajoutées à ST si la valeur courante de la variable *e* est à 1. Dans ce cas, elles mettent la valeur de cette variable à 0.

Enfin, le troisième pas de la procédure consiste à ajouter l'état st en cours, dans la liste des états bloquants, s'il n'a aucune transition sortante.

Quand *Ncou = N-1*, les états et les régions amont de la transition st sont sauvegardés dans l'ensemble des états marqués EM, alors que les régions aval sont regroupées dans l'ensemble des régions développées RD. Ces sauvegardes évitent des développements au prochain pas de calcul.

B) Sauvegarder les états bloquants dans BLOC.

C) Développement d'une région quelconque

Développer chacune des régions aval selon un principe similaire à ❶, ces nouvelles régions appartiennent alors à la fenêtre initiale d'intersection partielle f_0.

Cette étape procède au développement d'une région r_c à partir de son premier état, généré lors du pas ❶. L'algorithme correspondant est identique à celui utilisé pour définir la région initiale. Cependant, les états aval des transitions créées dans ce cas peuvent éventuellement appartenir à une région existante, autre que la région étudiée. Dans ce cas, les deux régions sont fusionnées en r_c car elles correspondent à une même situation de la commande.

Ncou = 0
❶ /* Développer la région initiale */
∀st = (q, y_0, E_0)∈ r_0 tel que $r_0 \in f_0$, ∀(q, Σ, q') ∈ Δ tel que (Σ=↑z ∧ z∈Z_{y0}) Faire:
i) ST = ST ∪ st' avec st'= (q', y_0, E_0); ii) TR = TR ∪ { (st, Σ, st') }; iii) $r_0 = r_0 \cup \{st'\}$; iv) $f_0 = f_0 \cup r_0$;
∀st = (q, y_0, E_0)∈ r_0 tel que $r_0 \in f_0$, ∀(q, Σ, q') ∈ Δ tel que (Σ=↓z ∧ z∉Z_{y0}) Faire:
i) ST = ST ∪ st' avec st'= (q', y_0, E_0); ii) TR = TR ∪ { (st, Σ, st') }; iii) $r_0 = r_0 \cup \{st'\}$; iv) $f_0 = f_0 \cup r_0$;
❷ Tant que Ncou ≤ N Faire :
A) /* Développer les transitions inter-régions supportées par les événements Σ_u et créer une nouvelle région */
∀st = (q, y, E) ∈ r_r Faire :
1- ∀ (q, ↑e, q') ∈Δ tel que (e∈E ∧ e=0) Faire :
- Si ∃ (((y, f(E), y') ∈ T) tel que f(E)=vrai) Alors st' = (q',y",E') ∧ (y"=y') Sinon st' = (q',y",E') ∧ (y"=y)
où la valeur de E' se distingue de E uniquement par la valeur de e qui devient 1 au lieu de 0 ;
- TR = TR ∪ { (st, ↑e, st') } ;
- Si st'∉ST Alors i) ST = ST ∪ {st'} ; ii) créer la région r = {st'} ⊂ ST ; iii) $f_0 = f_0 \cup r$;
- Si Ncou = N-1 Alors EM = EM ∪{st}, EM = EM ∪ {r_r}, RD=RD ∪ {r}; /* marquage pour l'intersection future */
2- ∀ (q, ↓e, q') ∈Δ tel que (e∈E ∧ e=1) Faire :
- Si ∃ (((y, f(E), y') ∈T) tel que f(E)=vrai) Alors st' = (q',y",E') ∧ (y"=y') Sinon st' = (q',y",E') ∧ (y"=y)
où la valeur de E' se distingue de E uniquement par la valeur de e qui devient 1 au lieu de 0 ;
- TR = TR ∪ { (st, ↓e, st') };
- Si st'∉ST Alors i) ST = ST ∪ {st'} ; ii) créer la région r = {st'} ⊂ ST ; iii) $f_0 = f_0 \cup r$;
- Si Ncou = N-1 Alors EM = EM ∪{st}, EM = EM∪{r_r}, RD = RD ∪ {r}; /* marquage pour l'intersection future */
B) Si ¬(∃(st, σ, st') ∈ TR) Alors BLOC = BLOC ∪ {st} /* Mémoriser les états bloquants */
C) /* Développer une région r_c quelconque telle que $f_0 = f_0 \cup r_c$ */
1- ∀(q, y_c, E_c)∈r_c tel que $r_c \in f_0$, ∀(q, Σ, q') ∈ Δ tel que (Σ=↑z ∧ z∈Z_{yc}) :
- Si st' ∉ ST avec st' = (q', y_c, E_c) Alors i) ST = ST ∪ {st'} ; ii) $r_c = r_c \cup \{st'\}$; iii) $f_0 = f_0 \cup r_c$;
- TR = TR ∪ { (st, Σ, st') };
2- ∀(q, y_c, E_c)∈r_c tel que $r_c \in f_0$, ∀(q, Σ, q') ∈ Δ tel que (Σ=↓z ∧ z∉Z_{yc}) :
- Si st' ∉ ST avec st' = (q', y_c, E_c) Alors i) ST = ST ∪ {st'} ; ii) $r_c = r_c \cup \{st'\}$; iii) $f_0 = f_0 \cup r_c$;
- TR = TR ∪ { (st, Σ,st') };
D) Ncou = Ncou + 1 /* Incrémentation du nombre Σ_u courant */

Figure 1. Développement de la fenêtre initiale f_0

➢ **Développement d'une fenêtre quelconque**

L'algorithme associé à l'étape 2 se décompose en 2 pas (figure 2.). Le premier pas (❶) consiste à récupérer les informations déjà développées dans l'intersection partielle précédente. Il s'agit de :

A) Récupérer les états et les régions marqués dans l'intersection précédente et appartenant à EM.

B) Récupérer les régions déjà développées regroupées dans RD.

C) Initialiser les ensembles EM et RD pour les futures sauvegardes à effectuer dans le développement d'une fenêtre f_c quelconque.

Le second pas (❷) correspond au ❷ de l'algorithme de développement de la fenêtre initiale (figure. 1).

Ncou = 0

❶ /* Récupération des informations calculées par l'intersection précédente*/

A)/* Récupération des états et des régions marqués */

- $\forall$ st = (q, y, E) $\in r_c$ tel que {st, r_c} $\in$ EM Faire : $f_c = f_c \cup r_c$;

B) /* Récupération des régions déjà développées */

- $\forall$(st, $\uparrow$e, st') $\in$ TR tel que (st $\in r_c$ et $r_c \in$ EM) et (st'=(q', y', E') $\in$ r et r $\in$ RD)
et E' se distingue de E uniquement par la valeur e qui devient 1 au lieu de 0
Faire : $f_c = f_c \cup r$;
- $\forall$(st, $\downarrow$e, st') $\in$ TR tel que (st $\in r_c$ et $r_c \in$ EM) et (st'=(q', y', E') $\in$ r et r $\in$ RD)
et E' se distingue de E uniquement par la valeur e qui devient 0 au lieu de 1
Faire : $f_c = f_c \cup r$;

C) EM = $\varnothing$ et RD = $\varnothing$ /* Initialisation des ensembles EM et RD */

❷ Tant que Ncou $\leq$ N Faire :

A) /*Développer les transitions inter-régions et créer une nouvelle région*/

B) /*Mémoriser les états bloquants */

C) /*Développer une région r_c quelconque telle que $f_c = f_c \cup r_c$ */

Figure.2. Développement d'une fenêtre quelconque f_c

Traitement des blocages et génération de l'automate de commande partielle

Par analogie avec la démarche de synthèse présentée dans la synthèse hors ligne, le principe de traitement des blocages consiste à retirer les états bloquants de $SYNC^N$ identifiés dans l'ensemble *BLOC*, ainsi que les transitions permettant d'atteindre ces états. Le retrait de ces évolutions bloquantes peut éventuellement générer une commande pour un nombre d'évolutions inférieur à *N*. Cette étape permet de générer l'automate partiel de commande OPT^N.

La réduction de la taille de l'automate de commande OPT^N est significative par rapport à une approche hors ligne. En effet, dans la synthèse hors ligne, la taille de l'automate *OPT* obtenu dans le cas le plus défavorable est proportionnelle à $|ASS|\times|SUP|\times 2^{\Sigma u/2}$. Or, dans le cas d'une synthèse en ligne, seul compte le nombre N de pas d'évolutions, déterminé pour l'intersection partielle; l'automate partiel obtenu comporte, dans le pire des cas, un nombre d'états proportionnel à $(|\Sigma_u|/2)^N$. Pour des systèmes de grande taille, le gain en espace mémoire est alors considérable. Concrètement, dès que le procédé génère une sortie, une nouvelle intersection partielle/réduction est développée pour un pas de plus, et l'automate est mis à jour. Par conséquent, l'architecture d'implantation repose sur l'exécution en parallèle de deux modules : un module de calcul en ligne de la commande qui effectue pendant l'exécution de la commande, un développement pour un pas de plus et un module d'exécution de la

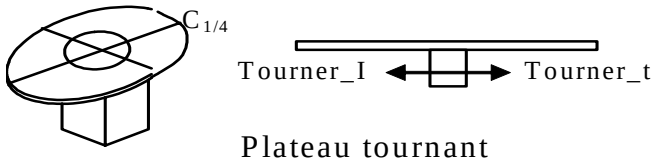
commande élaborée. Ainsi, le temps de calcul des automates $SYNC^N$ et de OPT^N influe le moins possible sur la réactivité de la commande.

Cependant, malgré l'optimisation de la taille mémoire, la synthèse en ligne ne permet pas de garantir l'absence de blocage lors du fonctionnement. En effet, étant donné qu'une intersection partielle ne prend en compte qu'une fenêtre limitée d'évolutions futures du système, des blocages non détectables dans la fenêtre courante peuvent se produire lors des intersections partielles suivantes. En augmentant la taille N de la fenêtre, ces blocages éventuels seront évités ou détectés suffisamment tôt pour pouvoir anticiper des actions de correction. Par conséquent, le choix de N constitue un compromis entre le risque de blocage et l'espace mémoire nécessaire à l'implantation de la commande élaborée.

ANNEXE.3

Exemple de Moteur électrique
Plateau tournant

Le plateau tournant effectue un mouvement de rotation dans le sens trigonométrique grâce à l'ordre TOURNER_T et dans l'autre sens grâce à l'ordre TOURNER_I. Le plateau est subdivisé en quatre sections, le capteur $C_{1/4}$ indique la rotation d'un quart de tour du plateau. Le plateau se trouve à l'état initial à la position pos_0.



Plateau tournant

1ère et 2ème et 3ème étapes : règles d'occurrence et relations de précédence et situation initiale

Conditions Initiales	Règles d'occurrence	Relations de précédence
$\downarrow TOURNER\_I$ $\downarrow TOURNER\_T$ $\uparrow pos_0$ $\downarrow c_{1/4}$	$\uparrow TOURNER\_I \rightarrow \downarrow pos_0$ $\uparrow TOURNER\_I \rightarrow \uparrow c_{1/4}$ $\uparrow TOURNER\_I \rightarrow \uparrow pos_0$ $\uparrow TOURNER\_I \rightarrow \downarrow c_{1/4}$	$\downarrow pos_0$ *précède* $\uparrow c_{1/4}$ $\downarrow c_{1/4}$ *précède* $\uparrow pos_0$
	$\uparrow TOURNER\_T \rightarrow \downarrow pos_0$ $\uparrow TOURNER\_T \rightarrow \uparrow c_{1/4}$ $\uparrow TOURNER\_T \rightarrow \uparrow pos_0$ $\uparrow TOURNER\_T \rightarrow \downarrow c_{1/4}$	$\downarrow pos_0$ *précède* $\uparrow c_{1/4}$ $\downarrow c_{1/4}$ *précède* $\uparrow pos_0$

Etape 4 : Construction des différents états (combinaisons):

états	*TOURNER_I*	*TOURNER_T*	Pos_0	$C_{1/4}$
0	0	0	0	0
1	0	0	0	1
2*	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

(*) État initial

Tableau 1 : tableau d'état du système (incohérences)

Etapes 5 :

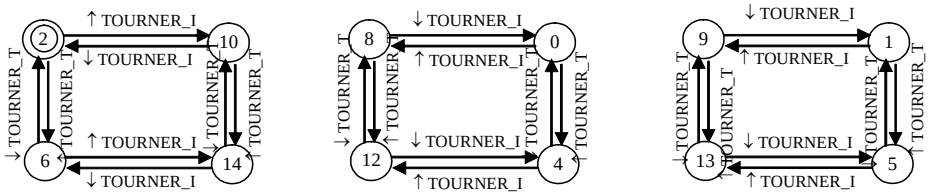
Supprimer les états (combinaisons) représentant les incohérences. Ici, c'est $pos_0 = c_{1/4} = 1$.

Etapes 6 :

Construction des évolutions commandables :

I/T \ **$Pos_0/c_{1/4}$**	**00**	**01**	**11**	**10**
00	0	1		❷
01	4	5		6
11	12	13		14
10	8	9		10

Diagramme des évolutions commandables



Construction de l'automate des transitions commandables.

Etape 7 : Construction des évolutions non commandables

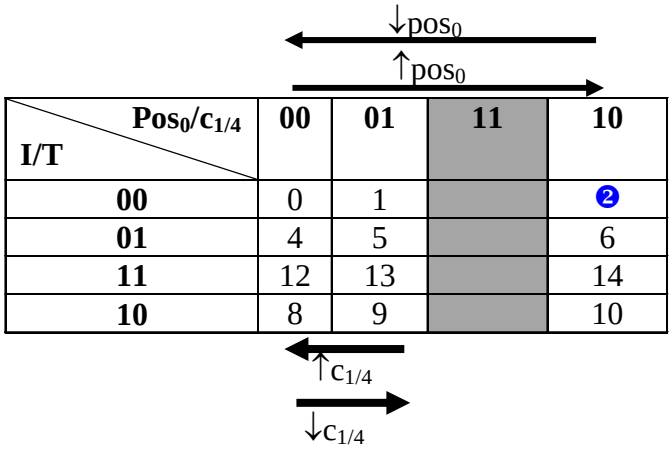
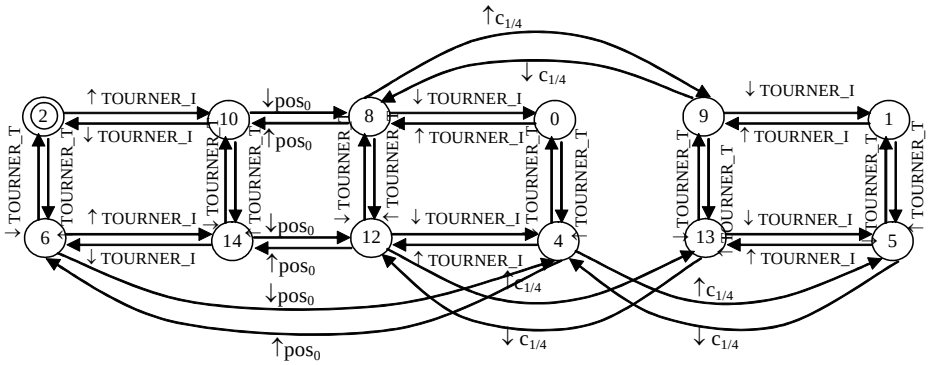


Diagramme des évolutions non commandables sans tenir des règles d'occurrence



Automate final de construction du mouvement du plateau tournant

Résumé :

La complexité croissante 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 à réaliser ces deux points, deux approches sont possibles : la première par validation et la seconde par synthèse.

Nous nous sommes intéressés dans ce mémoire à 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. Le travail de cette thèse constitue une démarche formelle permettant la synthèse de l'implantation la plus permissive de la commande à partir d'une spécification Grafcet, d'un système à commander et d'un ensemble de contraintes à respecter.

Pour établir cette synthèse, en plus de la spécification de la commande par Grafcet, il faut modéliser le système physique, ensuite, exprimer les contraintes de sûreté et de vivacité. Cependant, la modélisation du système physique et des contraintes s'avère être une tâche complexe. Pour pallier ces problèmes, notre démarche s'appuie sur 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 lui permettant de visualiser et d'analyser les séquences bloquantes ainsi que les corrections proposées.

Mots clés :
Systèmes à événements discrets, Grafcet, Synthèse de la commande, Modélisation, Partie Opérative, Contraintes, Logique booléenne.

Abstract:

The Automated Production Systems complexity, increases the requirements for users in term of operational safety as well as the program design reliability. To help the user to carry out these two points, two approaches are possible: the validation approach and the synthesis approach.

We are interested in this thesis, with the synthesis approach which consists in building a controller where the properties awaited for the system are taken into account from the beginning of the design. The work of this thesis concerns a formal approach allowing the synthesis of the most permissive implementation of an optimal control for a given Grafcet. It allows also a system to be controlled and a number of constraints to be respected.

To establish this synthesis, in addition to the specification of the Grafcet controller, it is necessary to model the physical system, then, to express the constraints of liveness and safety. However, modelling the physical system and the constraints is a complex task. To mitigate these problems, this approach is based on structured and advanced models while using the Boolean models containing rules for the plant and the logical equations in the traditional Boolean algebra for the constraints. A help in the development of the controller and the refinement of the starting models is proposed to the user enabling him to visualise and analyse the blocking sequences as well as the suggested corrections.

Key words:
Discrete event Systems, Grafcet, Synthesis of the controller, Modelling, Plant, Constraints, Boolean Logic.