Universidade do Minho
Escola de Engenharia

Ecole Normale Supérieure de Cachan
Laboratoire Universitaire de Recherche en Production Automatisée

José Mendes Machado

Influence de la prise en compte d'un modèle de processus en vérification formelle des Systèmes à Evénements Discrets

Maio 2006

Universidade do Minho
Escola de Engenharia

Ecole Normale Supérieure de Cachan
Laboratoire Universitaire de Recherche en Production Automatisée

José Mendes Machado

Influence de la prise en compte d'un modèle de processus en vérification formelle des Systèmes à Evénements Discrets

Tese de Doutoramento em
Engenharia Mecânica

Thése pour obtention du grade de Docteur de l'Ecole Normale Supérieure de Cachan
Spécialité: Électronique, Électrotecnhique et Automatique

Trabalho efectuado sob a orientação do
Professor Jaime C. L. Ferreira da Silva
(Universidade do Minho) e do
Professor Jean-Jacques Lesage
(École Normale Supérieure de Chachan)

Maio 2006

DECLARAÇÃO

Nome

José Mendes Machado

Endereço electrónico: *jmachado@dem.uminho.pt* Telefone: +351 253 510 227

Número do Bilhete de Identidade: 977 19 17

Título da tese :

Influence de la prise en compte d'un modèle de processus en vérification formelle des Systèmes à Evénements Discrets

Orientador(es):

Professor Jaime Carlos L. Ferreira da Silva (Universidade do Minho) e Professor Jean-Jacques Lesage (École Normale Supérieure de Cachan) Ano de conclusão: 2006

Designação do Ramo de Conhecimento do Doutoramento:

Doutoramento em Engenharia Mecânica

«É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE»

Universidade do Minho, 10 / 05 / 2006 .

Assinatura: _____

Ao meu filho José Augusto,

à minha esposa Maria dos Anjos,

aos meus pais

«I find the great thing in this world is not so much where we stand as in what direction we are moving.»

Oliver Wendell Holmes

Remerciements

Les travaux de recherche présentés dans ce mémoire ont été effectués dans le Laboratoire Universitaire de Recherche en Production Automatisé (LURPA) de l'École Normale Supérieure de Cachan (France) et dans le Département de Génie Mécanique (DEM) de l'Université du Minho (Portugal).

J'adresse mes sincères remerciements aux directeurs de thèse, Professeur Jean-Jacques Lesage du LURPA et Professeur Jaime Ferreira da Silva du DEM pour avoir assumé la direction de mes travaux.

Un merci spécial à Bruno Denis pour son encadrement, ses conseils techniques, sa patience ainsi que pour les heures qui a passé avec moi, pour le développement de ce travail.

Merci aux membres du jury pour le temps qu'ils m'ont consacré.

Merci à tous les membres du LURPA et du DEM pour avoir fait de ces années une période que je n'oublierai jamais.

Merci à mon fils José Augusto, mon épouse Maria dos Anjos et à mes parents pour avoir essayé de comprendre pourquoi je n'ai pas passé plus de temps avec eux durant ces quatre dernières années.

Merci à FCT (Fundação para a Ciência e Tecnologia), à PRODEP III (Programa de Desenvolvimento Educativo para Portugal) et au DEM pour le financement nécessaire pour que cette thèse ait pu se realiser avec succès.

«Influence de la prise en compte d'un modèle de processus en vérification formelle des Systèmes à Evénements Discrets»

Resumé

Ce travail est une thèse en co-tutelle encadrée par une convention de co-tutelle signée entre l'Université du Minho (UM) et l'École Normale Supérieure de Cachan (ENS). Le Département de Génie Mécanique (DEM) de l'UM et le Laboratoire Universitaire de Recherche en Production Automatisée (LURPA) de l'ENS de Cachan sont les principaux acteurs de cette coopération matérialisé par ce travail de thèse de l'auteur. Le travail de thèse a été élaboré dans les deux institutions et a été devélopé dans le domaine de la Sûreté de Fonctionnement de systèmes automatisés.

La Sûreté de Fonctionnement (SdF) fait aujourd'hui irruption dans le cahier des charges d'un nombre de systèmes automatisés beaucoup plus important que par le passé. Assurer la sûreté d'un système nécessite une approche globale (pour ne pas laisser de maillon faible) qui porte sur l'ensemble des activités d'ingénierie puis d'exploitation du système. Dans le domaine des Systèmes à Evénements Discrets (SED), parmi les récentes contributions scientifiques majeures à l'amélioration de la SdF des systèmes automatisés on peut citer : la synthèse de contrôleurs sûrs grâce à la théorie de supervision, le diagnostic qui consiste à identifier des comportements fautifs et la vérification formelle de contrôleurs qui prouve la véracité de propriétés attendues. Nous avons pour notre part considéré un SED comme étant composé d'un contrôleur et d'un processus en boucle fermée, tous deux aux comportements asynchrones, synchronisés lors des échanges d'informations (cadencés par les phases de lecture des entrées et d'affectation des sorties du contrôleur).

L'objectif principal de cette thèse est d'évaluer les apports, les limites et les complémentarités de la vérification du comportement d'un contrôleur logique par model-checking en prenant en compte, ou non, un modèle du processus commandé. Pour la modélisation nous proposons une classe d'automates non déterministes, communiquant par partage de variables logiques et aux transitions labellées par des expressions booléennes. Le modèle du processus est alors obtenu par coordination des comportements de modules élémentaires, grâce à un séquenceur qui permet une évolution asynchrone des modules jusqu'à atteindre une situation stable du modèle.

La vérification formelle d'un ensemble représentatif de propriétés est traitée sur un système automatisé servant d'exemple support. Nous généralisons ces résultats et présentons une approche mixant la vérification formelle avec et sans modèle de processus. Cette approche est basée sur une double typologie des propriétés et des variables utilisées pour les exprimer. On distingue d'une part les propriétés de vivacité des propriétés de sûreté, et d'autre part les propriétés exprimées à partir de variables contrôlables ou non. Cette approche couplée propose un cadre de vérification plus rigoureux et plus sûr que celui de l'utilisation la plus souvent pratiquée, sans aucun modèle de processus.

«Influência da Definição do Modelo de Análise de Processo na verificação formal de Sistemas de Manufactura a Eventos Discretos»

Resumo

Este trabalho consiste na realização de uma tese em co-tutela, enquadrada por uma convenção de co-tutela celebrada entre a Universidade do Minho (UM) e a École Normale Supérieure (ENS) de Cachan. O Departamento de Engenharia Mecânica da UM e o Laboratoire Universitaire de Recherche en Production Automatisé da ENS de Cachan são os principais actores desta colaboração materializada por esta tese de doutoramento do autor. O trabalho de tese foi elaborado nas duas instituições e foi desenvolvido no domínio da Segurança de Funcionamento de sistemas automatizados.

A Segurança de Funcionamento (SdF) aparece, nos dias de hoje, tratada nos cadernos de encargos de cada vez mais sistemas automatizados de uma forma mais importante do que no passado. Garantir a segurança de um sistema implica uma abordagem global (para que nenhum aspecto seja descurado) que considera as várias vertentes da actividade de engenharia e posteriormente sobre a colocação em funcionamento e exploração do sistema automatizado. No domínio dos Sistemas a Eventos Discretos (SED), de entre as várias contribuições científicas de maior relevo para a melhoria da SdF de sistemas automatizados, pode-se citar: a síntese de controladores seguros a partir da aplicação da teoria da supervisão, o diagnóstico que consiste em identificar os comportamentos de falha e a verificação formal de controladores que prova a veracidade de certas propriedades comportamentais esperadas. Neste caso específico, foi considerado um SED como sendo composto por um controlador e um processo em malha fechada, cada um deles com comportamentos assíncronos, sendo sincronizados através da troca de informações entre eles (cuja cadência é caracterizada pelas fases de leitura das entradas e afectação das saídas do controlador).

O objectivo principal desta tese é de avaliar as vantagens, os limites e a complementaridade na verificação formal de um controlador lógico, por model-checking, considerando, ou não, um modelo do processo comandado. Para a modelização propõe-se uma classe de autómatos não determinísticos, que comunicam por partilha de variáveis lógicas e com transições «labelizadas» por expressões booleanas. O modelo do processo é obtido pela coordenação dos comportamentos de módulos elementares graças a um sequenciador que permite uma evolução assíncrona dos módulos até que se atinja uma situação estável do modelo.

A verificação formal de um exemplo representativo de propriedades é tratado num sistema automatizado que serve de exemplo suporte. Generaliza-se os resultados obtidos e apresenta-se uma abordagem onde é misturada a verificação formal com e sem modelo do processo. Esta abordagem é baseada numa dupla tipologia das propriedades e das variáveis utilizadas para as exprimir. Distingue-se, por um lado, as propriedades de vivacidade e de segurança e, por outro, as propriedades exprimidas a partir de variáveis controláveis ou não. Esta abordagem «mista» propõe um quadro de verificação formal mais rigoroso e mais seguro do que é usualmente praticado sem a utilização de qualquer modelo do processo.

Table des matières

Remerciements v
Resumé vii
Resumo ix
Table des matières xi

Introduction 1

Chapitre 1 La prise en compte d'un modèle de processus en conception des SED 5

1.1 Pourquoi un modèle du processus (objectifs, possibilités et limites) ? 7
1.1.1 Synthèse d'un contrôleur 7
1.1.2 Analyse 8
1.1.3 Identification et test 10
1.1.4 Bilan et limites 11
1.2 Comment obtenir le modèle du processus ? 12
1.2.1 Approche monolithique 12
1.2.2 Approches modulaires 12
1.2.3 Bilan 13
1.3 Le cas de la vérification formelle par model-checking 13
1.3.1 Exemple illustratif 13
1.3.2 Preuve des propriétés 14
1.3.3 Typologie et expression de propriétés 15
1.3.4 Granularité du modèle du processus 16
1.3.5 Formalisme à utiliser 16
1.3.6 Couplage des modèles du contrôleur et du processus 17
1.3.7 Bilan sur la vérification par Model-Checking 18
1.4 Bilan du chapitre 1 18

Chapitre 2 Approche globale, hypothèses et choix retenus 21
2.1 Approche globale retenue 23
2.2 Hypothèses retenues 24
2.2.1 Hypothèses liées aux propriétés 25
2.2.2 Hypothèses concernant le model-checking 25
2.2.3 Hypothèses liées aux inter-actions contrôleur-processus 26
2.3 Formalisme de modélisation retenu 27
2.3.1 Caractéristiques attendues 27
2.3.2 Définition informelle de la classe d'automates retenue 28
2.3.3 Définition formelle de la classe d'automates retenue 28
2.3.4 Règles d'évolution 29
2.3.5 Notations et représentation graphique 29
2.4 Bilan du chapitre 2 31

Chapitre 3 Modélisation du processus 33
3.1 Notion de modules 35
3.1.1 Granularité de structuration 35
3.1.2 Granularité de comportement 37
3.1.3 Modélisation de plusieurs chaînes fonctionnelles utilisant des composants communs 41
3.1.4 Modélisation du temps logique 43
3.2 Association de modules 44
3.2.1 Séquenceur du processus 45
3.2.2 Communication entre séquenceur et modules du processus 46
3.2.3 Stabilité du modèle du processus 47
3.2.4 Synchronisation des modèles de temporisation avec le séquenceur du processus . 49
3.3 Modèle complet d'un processus composé d'une chaîne fonctionnelle 52
3.4 Bilan du chapitre 3 52

Chapitre 4 Construction du modèle de vérification du système complet 55
4.1 Modèle du contrôleur et de son séquenceur 57
4.2 Modèle des propriétés et de leur séquenceur 62
4.3 Séquenceur général 65
4.4 Présentation de l'exemple support 68
4.5 Modèle global du système 71
4.5.1 Modèle du contrôleur 71
4.5.2 Modèle du processus 73
4.5.3 Modèle des propriétés à prouver 85
4.5.4 Séquenceur général 89
4.6 Bilan du chapitre 4 89

Chapitre 5 Étude comparée de la vérification formelle avec ou sans modèle de processus 91

5.1 Traduction du modèle en code d'entrée du model-checker NuSMV 93
5.1.1 Règle 1 : Structure générale du code d'entrée de NuSMV 93
5.1.2 Règle 2 : Traduction de la structure d'un module du modèle 93
5.1.3 Règle 3 : Traduction des assignations des variables du modèle 94
5.1.4 Règle 4 : Gestion de la simultanéité de franchissement des transitions 95
5.1.5 Règle 5 : Traduction des propriétés en code NuSMV 95
5.1.6 Algorithme de génération du code NuSMV 96

5.2 Vérification des propriétés avec les deux approches NMB et MB. 96

5.3 Interprétation des résultats obtenus 97
5.3.1 Remarques générales 97
5.3.2 Remarques détaillées par propriété 98
5.3.3 Remarques sur les deux approches : NMB et MB 101
5.3.4 Remarques sur la spécification du contrôleur de l'étude de cas 102

5.4 Proposition d'une approche mixte NMB-MB 104
5.4.1 Propriétés de sûreté portant sur des variables observables 104
5.4.2 Propriétés de sûreté portant sur au moins une variable non observable 104
5.4.3 Propriétés de vivacité 104
5.4.4 Arbre de décision selon la propriété 105

5.5 Bilan du chapitre 5 106

Conclusions et perspectives 107

Références bibliographiques 109

Références techniques 117

Annexe 1 Modèle complet de l'exemple support utilisé 119

A1.1 Structure générique des modèles 121

A1.2 Modèle complet pour l'approche model-based 124
A1.2.1 Valeur initiale des variables du modèle 124
A1.2.2 Séquenceur général 125
A1.2.3 Modèle du contrôleur 126
A1.2.4 Modèle du processus 127
A1.2.5 Modèle des propriétés 134

Annexe 2 Code NuSMV complet de l'exemple support utilisé 137

A2.1 Fichier «exemple_support-specif_OK_MB.nusmv» 138

A2.2 Sorties de la commande Unix «time NuSMV -r -reorder -dynamic exemple_support-specif_OK_MB.nusmv» 158

Introduction

La Sûreté de Fonctionnement (SdF) fait aujourd'hui irruption dans le cahier des charges d'un nombre de systèmes automatisés beaucoup plus important que par le passé. Plusieurs raisons convergentes permettent d'expliquer cet «engouement» pour la SdF. En tout premier lieu, les systèmes automatisés sont désormais omniprésents dans notre vie quotidienne et les fonctions qu'ils assurent sont de plus en plus sophistiquées. Par contre, l'utilisation de systèmes «grand public» tels que les distributeurs automatiques de billets ou les régulateurs de vitesse automobile ne doit requérir aucune compétence technique particulière. Ces systèmes doivent donc être *sûrs*, c'est-à-dire fiables, disponibles et tolérants aux erreurs de manipulation. Par ailleurs, l'évolution des préoccupations et des priorités sociétales, en mettant au centre de nos préoccupations l'exigence de traçabilité, le respect de l'environnement, ou le principe de précaution par exemple, font considérablement évoluer notre propre perception de la sûreté. C'est ainsi que la notion même de système critique (ou de fonction critique d'un système) s'est aujourd'hui étendue à de nombreux domaines dans lesquels elle était absente (agroalimentaire, énergie, santé, traitement de l'eau, des déchets, ...). Enfin, le niveau d'exigence du consommateur croît sans cesse alors que les marchés sont de plus en plus concurrentiels. Il s'agit donc pour l'industrie de proposer en permanence des produits innovants, aux fonctions sophistiquées, bon marché et sûrs de fonctionnement (on peut citer à ce sujet les fonctions d'assistance à la conduite automobile qui, après avoir été introduites pour améliorer la sécurité des passagers, comme l'Anti-Blocking System (ABS) ou l'Electronic Stability Program (ESP), se prolongent aujourd'hui par l'automatisation de multiples fonctions de confort comme le passage automatique des vitesses «Shift-by-wire»).

La sûreté de fonctionnement a pour objectif global de répondre aux exigences de fiabilité, de disponibilité et de maintenabilité des systèmes. Par son impact direct sur la sécurité des personnes et des biens, la fiabilité des systèmes critiques (transport, espace, nucléaire, ...) a depuis longtemps mobilisé les efforts de la communauté scientifique. Assurer la sûreté d'un système nécessite une approche globale (pour ce pas laisser de maillon faible) qui englobe l'ensemble des activités d'ingénierie, puis, après la mise en service, l'ensemble des activités d'exploitation et de maintien en conditions d'exploitation opérationnelle de ce système. C'est pourquoi la communauté scientifique s'est employée à procurer aux ingénieurs des modèles, des méthodes et des outils ayant comme objectifs complémentaires de prévoir les dysfonctionnements (fiabilité prévisionnelle, AMDEC, ...), de tolérer des dysfonctionnements (redondance, modes dégradés ou reconfiguration, ...), ou de démontrer l'obtention d'un niveau requis de SdF (spécifications formelles prouvables, crash tests, ...).

Dans le domaine des Systèmes à Evénements Discrets (SED), parmi les contributions scientifiques majeures à l'amélioration de la SdF des systèmes de production de ces dernières années on peut citer :

- la *synthèse de contrôleur* sûrs grâce à la théorie de supervision. Cette approche consiste à restreindre le comportement du système automatisé par le biais d'un superviseur, de manière à ce que le contrôleur ne génère pas de commande conduisant à un dysfonctionnement (ce que le contrôleur initial permettait le plus souvent) ;
- le *diagnostic*, qui consiste à identifier des comportements fautifs non directement observables durant le fonctionnement du système. Ces événements non observables sont déterminés à partir d'un modèle du système et des séquences d'événements observés grâce aux capteurs ;
- la *vérification formelle de contrôleurs*, qui consiste à prouver que ceux-ci vérifient un ensemble de propriétés de bon fonctionnement attendues et qu'en aucune manière ils ne peuvent générer de comportements fautifs.

Ces trois domaines de recherche sont bien entendu largement complémentaires mais dans cette thèse nous nous intéressons spécifiquement à l'amélioration de la sûreté de fonctionnement grâce à la vérification formelle du contrôleur ; nous allons maintenant préciser notre contribution.

Ce n'est que relativement récemment que les techniques de vérification formelle, issues de l'informatique, et plus particulièrement le model-checking, ont été portées et appliquées à la vérification des SED. Il s'agit aujourd'hui d'un domaine de recherche mature dans lequel de nombreux et importants résultats théoriques ont été obtenus depuis une dizaine d'années. La confrontation de ces résultats aux contraintes, aux besoins et aux pratiques de l'industrie fait cependant émerger de nombreux problèmes de nature méthodologique. Parmi ces problèmes, le choix de procéder à une vérification des propriétés du contrôleur en prenant en compte ou non un modèle du processus commandé est souvent difficile. Nous étudions dans ce mémoire les vertus comparées de ces deux approches en terme d'efficience de la vérification (relation entre la pertinence des résultats obtenus et l'effort de modélisation et de calcul nécessaire pour obtenir ces résultats). Dans le cas d'une vérification où le modèle du contrôleur est couplé à un modèle du processus commandé, se pose naturellement le problème de construction du modèle du processus. Il s'agit en effet d'une tâche difficile à laquelle pourtant peu de travaux de recherche ont été dédiés. Nous proposons également une technique de construction de modèles comportementaux pour les systèmes commandés industriels, en vue d'une vérification du contrôleur. Nous avons pour cela retenu une approche modulaire et nous utilisons une classe appropriée d'automates à états finis. Pour obtenir le modèle global du système commandé, les évolutions parallèles des automates élémentaires sont coordonnées par un «séquenceur» qui garantit la cohérence de ces évolutions. Un même exemple d'application est utilisé à la fois pour illustrer la construction d'un tel modèle de processus et pour étudier l'intérêt du couplage entre un modèle du contrôleur et un modèle du processus en vérification par model-checking.

De manière à présenter notre approche et nos résultats, nous avons organisé ce mémoire comme suit :

- dans le Chapitre 1 nous présentons au travers d'une revue de littérature l'utilisation d'un modèle du processus commandé dans les différentes techniques utilisées pour améliorer de sûreté de fonctionnement des SED : synthèse, diagnostic et vérification formelle. Puisque notre travail concerne le domaine de la vérification formelle, nous développerons plus particulièrement ce domaine ;
- dans le Chapitre 2 nous ferons une description générale de l'ensemble de la démarche que nous avons adoptée. Nous préciserons également les hypothèses que nous avons retenues et définirons la classe d'automates à états finis que nous avons choisi pour la modélisation comportementale ;

- le Chapitre 3 sera entièrement consacré à la modélisation du processus. Nous avons retenu un approche modulaire, bien adaptée à la modélisation des SED complexes, qui nécessite la synchronisation locale de comportements parallèles décrits par des modules. La construction de ces «séquenceurs» sera tout particulièrement développée ;
- dans le Chapitre 4 nous présenterons le cadre formel pour la modélisation d'un système automatisée complet «prêt à être vérifié» (composé par le contrôleur, le processus commandé et les propriétés à vérifier). Nous utiliserons un exemple nous permettant à la fois d'illustrer notre démarche et de mettre en évidence quelques aspects importants de cette modélisation ;
- le Chapitre 5 sera consacré à la vérification formelle, par model-checking, de l'exemple traité dans le chapitre 4. Tous les aspects de traduction du modèle global établi précédemment dans le code d'entrée du model-checker seront présentés. Les résultats obtenus seront également discutés et généralisés dans ce chapitre. Finalement on présentera une stratégie de vérification conduisant à l'établissement de preuves «fortes» ;
- finalement, après avoir présenté une synthèse des résultats obtenus, nous conclurons et dévoilerons quelques pistes pour des travaux futurs prolongeant ce travail de thèse.

Chapitre 1
La prise en compte d'un modèle de processus en conception des SED

La prise en compte d'un modèle de processus en conception des SED peut avoir différents objectifs. Les modèles de processus concernent en effet aussi bien la synthèse, la vérification formelle, la simulation que l'identification de contrôleurs. Dans ce chapitre, une analyse bibliographique sera présentée sur les travaux qui prennent en compte le modèle du processus, et on discutera de l'intérêt mais aussi des limites de cette utilisation. Dans notre travail, on considère les modèles de processus dans les activités de vérification formelle par model-checking, et notre objectif est d'augmenter la confiance que l'on a dans les programmes de contrôleurs. On va montrer qu'il y a des cas où l'utilisation d'un modèle de processus est nécessaire pour la preuve de certaines propriétés. En revanche l'utilisation d'un tel modèle doit faire l'objet d'une attention particulière quant à sa granularité, sa syntaxe, où à l'approche méthodologique retenue pour le construire.

1.1 Pourquoi un modèle du processus (objectifs, possibilités et limites) ?

Dans cette section il est présenté une revue bibliographique sur les différentes applications et les différents contextes d'utilisation d'un modèle du processus. On peut d'ores et déjà noter que dans les SED, la sûreté de fonctionnement est souvent à l'origine de l'usage de modèles de processus.

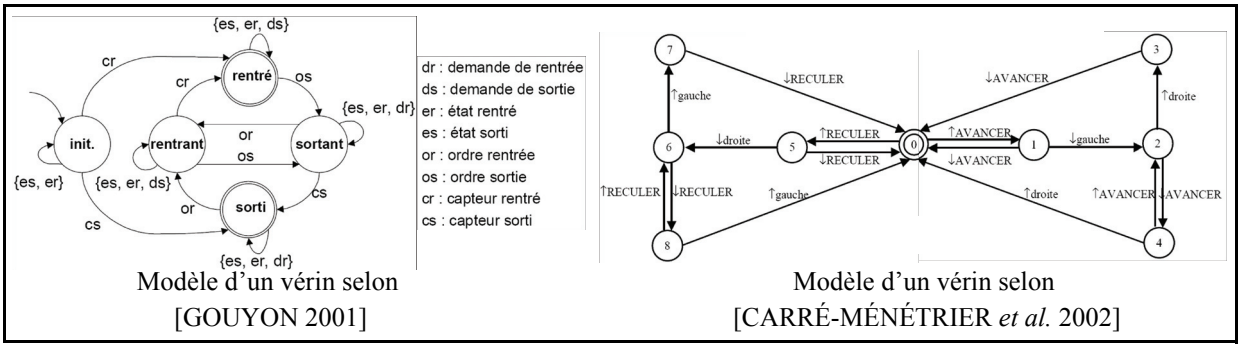
1.1.1 Synthèse d'un contrôleur

La synthèse de contrôleurs (superviseurs) des systèmes à événements discrets (SED) trouve son origine dans la théorie de Supervision des SED, initiée par Ramadge et Wonham [WONHAM & RAMADGE 1987], développée dans [RAMADGE & WONHAM 1987] [RAMADGE & WONHAM 1989], et ultérieurement facilitée dans sa mise en application pratique par [KUMAR 1991]. Cette théorie est basée sur le principe de la séparation des concepts de fonctionnement du processus en boucle ouverte et celui du contrôleur en boucle fermée. Son objectif principal est d'établir, pour les SED, des principes analogues à ceux de la théorie de la commande des systèmes continus, tels que les notions de commandabilité, d'observabilité, en employant des techniques basées sur les langages formels et les automates. Ceci doit permettre d'évaluer différentes lois de commande et d'effectuer la synthèse de la stratégie de commande la plus permissive possible par rapport aux spécifications.

Le problème associé à cette théorie est que la sémantique des modèles du processus et du superviseur n'est pas adaptée à la plupart des systèmes commandés en temps réel [GOUYON *et al.* 2003] [PHILIPPOT *et al.* 2003]. En effet, la théorie de supervision stipule que le processus est un générateur spontané d'événements, classés en événements contrôlables «E*c*» et non contrôlables «E*u*». L'action du contrôleur, appelée supervision, consiste uniquement à autoriser ou inhiber les événements contrôlables afin de restreindre le comportement en boucle fermée du processus aux séquences d'événements maximales admissibles par rapport au comportement désiré. Or les événements, dans les processus réels, ne sont pas toujours générés de façon spontanée, mais réagissent à des commandes d'entrée, générant des réponses en sortie.

Dans le cadre de la théorie de supervision des SED, la synthèse du contrôleur consiste à déduire, pour un processus donné, le superviseur décrivant le comportement contrôlable non-bloquant le plus permissif possible par rapport aux contraintes de sécurité et de vivacité spécifiées. Un comportement est dit contrôlable lorsqu'aucune occurrence d'événement non-contrôlable (physiquement possible) n'entraîne une évolution vers un état n'appartenant pas à ce comportement. Le modèle du processus, conjointement avec quelques spécifications prétendues pour le fonctionnement du système, est utilisé pour la synthèse d'un superviseur qui fait en sorte que le comportement du processus n'évolue que pour des situations comportementales désirées et non pour des situations dangereuses ou non désirées. Cette supervision n'est possible que parce que le comportement réel du processus est connu. L'utilisation d'un modèle du processus est dans ce cas strictement indispensable.

La «qualité» du contrôleur obtenu dépend directement de la «qualité» du modèle du processus adopté. Ses caractéristiques, comme par exemple la granularité (voir "Granularité du modèle du processus", page 16), sont fondamentales pour la qualité et la taille du contrôleur obtenu. Par exemple, dans les travaux de [GOUYON 2001] et de [CARRÉ-MÉNÉTRIER *et al.* 2002], il y a deux modèles différents pour un vérin (5 états dans le premier et 9 états dans le deuxième), figure 1. Sans une analyse détaillée de ces modèles, on peut facilement constater que dans les deux cas le superviseur obtenu sera différent.



Modèle d'un vérin selon [GOUYON 2001]

Modèle d'un vérin selon [CARRÉ-MÉNÉTRIER *et al.* 2002]

Figure 1: Deux approches différentes pour un modèle de vérin.

Le processus est fréquemment modélisé par des automates à états modulaires de petite taille (quelques états), où chacun d'eux décrit un sous-ensemble du processus [CARRÉ-MÉNÉTRIER *et al.* 2002] [GOUYON 2001]. Certains travaux proposent une modélisation du processus à l'aide d'autres formalismes, comme par exemple dans [ALPAN & JAFARI 2002] où tout le système (contrôleur et processus) est représenté par des réseaux de Petri.

En synthèse, si le modèle de processus est erroné ou incomplet, le contrôleur obtenu sera lui aussi erroné ou incomplet, puisque son obtention dépend directement du modèle du processus. Cependant, bien que la théorie de supervision soit très sensible à la qualité des modèles de processus, peu de travaux portent sur la pertinence de ces modèles ni sur la façon de les élaborer.

1.1.2 Analyse

L'objectif des techniques d'analyse en conception des SED est de réduire ou d'éliminer les erreurs dans les programmes de commande dans la perspective d'obtentir un fonctionnement plus sûr du système automatisé. Les deux techniques courantes d'analyse sont la simulation et la vérification formelle.

1.1.2.1 Simulation

La simulation est une technique qui permet d'expérimenter un nombre fini et relativement réduit de possibilités d'évolution d'un système automatisé. Donc les résultats qui peuvent être obtenus ne sont valables que pour certains scenarii d'évolution. Dans les nombreux travaux développés en simulation il est usuel de considérer, d'une façon plus ou moins élaborée, le modèle du processus, comme par exemple :

- [BARESI *et al.* 1998] [BARESI *et al.* 2000] où le processus est modélisé par blocs Simulink,
- [AMERONGEN 2003] où le processus est modélisé par bond graphs,
- [BARTON 1992] où le processus est décomposé en entités et chaque entité est modélisée par un automate fini,
- [MATTSSON *et al.* 1998] [ELMQVIST *et al.* 1999] où le processus est modélisé par le langage *Modelica*,
- [LIU, 2004] où il est proposé une méthodologie pour modélisation du processus, SPOA (System Performance-Oriented Automation).

Dans tous ces travaux, le modèle de processus est généralement utilisé pour obtenir des scénarios réalistes d'excitation des modèles de commande. Plutôt que de générer hors ligne les évolutions de nombreuses entrées logiques du système, il est préférable de les obtenir à partir de l'évolution d'un modèle de processus. Dans ce cas le modèle de processus à une influence directe

sur pertinence des stimuli du modèle, et donc sur le crédit que l'on pourra donner aux résultats de simulation.

1.1.2.2 Vérification formelle

L'utilisation de méthodes formelles dans la vérification des contrôleurs des SED peut être faite ou être classée, selon trois critères différents [FREY & LITZ 2000-a] :

- La méthode utilisée : «Model-checking» [LAMPERIERE-COUFFIN *et al.* 1999], «Theorem-proving» [ROUSSEL & DENIS 2002], «Reachability analysis» [FREY & LITZ 2000-b].
- Le formalisme adopté : Réseaux de petri [MERTKE & MENZEL 2000], Systèmes Condition/Événement [RAUSCH & KROGH 1998], Automates à états [HASSAPIS *et al.* 1998].
- L'utilisation ou non d'un modèle de processus :
  - «Non model-based» sans considération d'un modèle de processus [PROBST *et al.* 1997] [DE SMET & ROSSI 2002] ;
  - «Constrained-based» en considérant un modèle rudimentaire traduit par certaines contraintes [CANET *et al.* 2000];
  - «Model-based» avec un vrai modèle du processus, élaboré au travers d'un formalisme bien défini [KOWALEWSKI & PREUßIG 1996]. Ce modèle peut être plus ou moins raffiné, selon les propriétés qu'on souhaite prouver.

La vérification formelle est une technique exhaustive (preuve) mais elle est difficile à mettre en oeuvre et elle n'aboutie pas toujours à une solution.

Etudions plus en détail l'utilisation de modèles du processus dans les méthodes de vérification formelle de contrôleurs. Tout système automatisé est composé d'un contrôleur et d'un processus. Les sorties du contrôleur sont les entrées du processus tandis que les sorties du processus sont les entrées du contrôleur (figure 2).

Beaucoup de travaux ont abordé la vérification formelle des contrôleurs logiques. Parmi les plus significatifs on peut citer [BORNOT *et al.* 2000] [LAMPERIERE-COUFFIN *et al.* 1999] [MERTKE & FREY 2001] [DE SMET & ROSSI 2002] [FREY & LITZ 2000-b] [ROSSI 2004]. Il y a également des travaux qui sans utiliser un modèle explicite du processus, introduisent des contraintes de comportement dues au processus, comme par exemple dans [CANET *et al.* 2000]. Dans ce cas l'inclusion de certaines contraintes améliore les résultats de vérification obtenus.

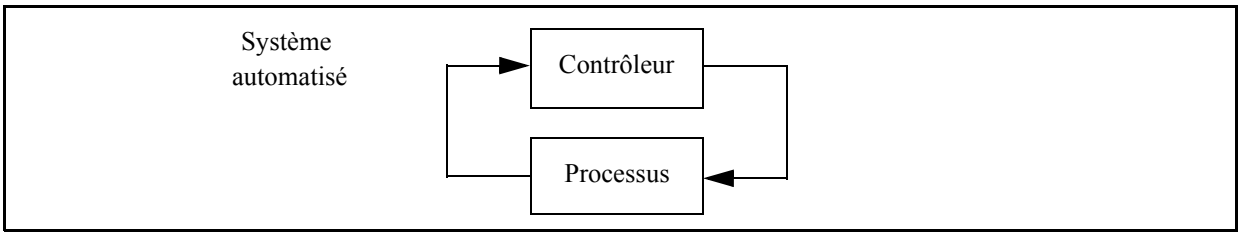


Figure 2: Système automatisé composé d'un Contrôleur et de Processus.

Si nous considérons des travaux pour lesquels le processus a été considéré explicitement, nous pouvons citer [RAUSCH & KROGH 1998] [HASSAPIS *et al.* 1998] [KOWALEWSKI & PREUßIG 1996] [MERTKE & MENZEL 2000] où le processus est modélisé à l'aide des formalismes suivants : Réseaux de Petri, Automates à états, Systèmes Condition/Événement. En général, la modélisation du processus est faite de façon monolithique (en un seul bloc) et les processus modélisés sont de taille réduite. Dans les cas où la modélisation a été abordée de façon modulaire [ZAYTOON & CARRÉ-MÉNÉTRIER 1999], le modèle global du processus est le plus générale-

ment obtenu par le produit cartésien de ses modules constitutifs. Ce modèle global est, même pour une étude de cas relativement simple, très complexe et de dimension (en nombre d'états et transitions) importante qui inclut de surcroît des états qui n'ont pas toujours de réalité physique. Dans les travaux analysés, quand le modèle du processus est utilisé, c'est pour réduire le nombre d'états atteignables du contrôleur et pour permettre l'obtention d'une réponse (sur la vérification formelle d'une propriété) quand cette réponse n'est pas obtenue sans l'utilisation d'un modèle du processus. À l'heure actuelle, les modèles du processus utilisés en vérification formelle ne sont souvent vus que comme un simple complément pour améliorer les performances des techniques de vérification formelle comme les model-checkers.

Un autre facteur à prendre en compte est le niveau de détail du modèle du processus utilisé (granularité), car ce facteur impacte directement la dimension du modèle global du système (plus grand nombre d'états, avec une influence directe sur le temps nécessaire à l'obtention d'une réponse) et sur le type de propriétés à prouver.

La modélisation des interactions entre le comportement du contrôleur et celui du processus n'est généralement pas considérée. Dans tous les travaux qui utilisent un modèle du processus il n'y a pas une réelle prise en compte des aspects technologiques du contrôleur et du processus, notamment de leurs caractéristiques physiques et comportementales. En effet, leurs évolutions dans le temps sont asynchrones mais avec des instants de resynchronisation (lecture des informations capteur, mise à jour des sorties du contrôleur), et cela avec des échelles de temps complètement différentes entre un contrôleur réactif et un processus physique plus lent. Ces aspects, importants en vérification formelle, seront traités en détail dans la section "Le cas de la vérification formelle par model-checking", page 13.

1.1.3 Identification et test

L'identification est une technique d'étude des SED basée sur un concept de base : l'observabilité [LI & WONHAM 1988]. Cette technique peut être appliquée à la validation de l'implantation d'une spécification (test), au reverse-engineering d'installations opérationnelles, ou à l'obtention d'un modèle de comportement pour assurer un diagnostic en ligne par le modèle. Dans le cas d'un système automatisé incluant à la fois un contrôleur et un processus, deux modes d'identification peuvent être envisagés :

- à partir d'observations du contrôleur *in situ* interagissant avec le processus réel comme présenté figure 3(a) (solution largement utilisée dans les techniques de diagnostic),
- à partir d'observations du contrôleur *libre* ou en *«boucle ouverte»* réagissant aux changements de valeur de ses entrées qui sont excitées spécifiquement pour l'identification, comme présenté figure 3(b) (solution privilégiée pour les activités de test).

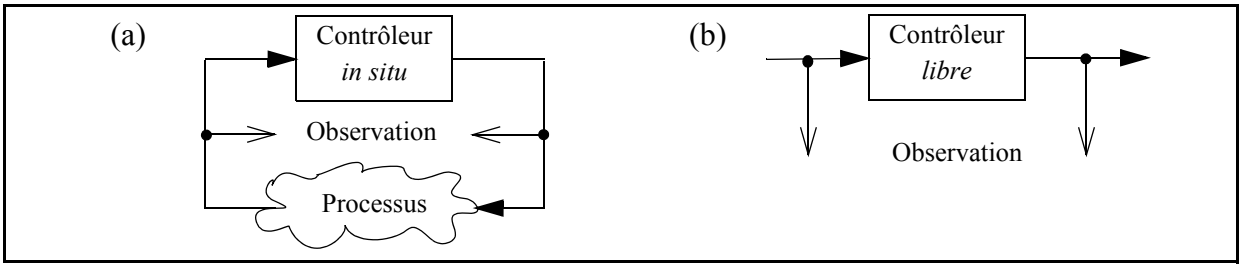


Figure 3: Modes d'identification : (a) avec le processus ; (b) sans le processus.

Dans le domaine des systèmes à événements discrets, le formalisme classiquement retenu pour l'identification est la machine à états finis [KLEIN 2005], ou ses extensions comme par exemple les réseaux de Petri [MEDA-CAMPAÑA & LÓPEZ-MELLADO 2002].

Dans le cadre du diagnostic «modèle based» qui consiste à identifier la divergence entre les signaux observés sur le système en fonctionnement avec les signaux prévus par un modèle de comportement [DAVIS & HAMSCHER 1988], l'identification est une approche expérimentale qui permet d'obtenir le modèle du système. Le modèle de comportement du système peut être un modèle de bon fonctionnement ou un modèle incluant des défaillances [SZTIPANOVITS & MISRA 1994].

Pour l'obtention des modèles de comportement du système deux autres possibilités se dégagent : la construction des modèles basés dans la connaissance du fonctionnement du système [LAFORTUNE et al. 2001] («knowledge-based models») et la construction du modèle à travers la collecte de l'historique des données produites pendant le fonctionnement du système («data-driven models») [PAPADOPOULOS & MCDERMID 2001], [KLEIN 2005].

1.1.4 Bilan et limites

Dans les travaux identifiés, la modélisation du processus est toujours réalisée par des systèmes Condition/Evénement, des automates à états finis, des automates hybrides, des réseaux de Petri (formalismes classiques), ou par des schémas blocs ou des bond graphs (formalismes utilisés plus récemment pour la modélisation du processus). Le choix de la technique de modélisation reste principalement dicté par l'usage qui en sera fait et non pas par les caractéristiques propres du processus.

La praticabilité des approches est souvent une limitation à la prise en compte de processus de taille significative. Les modèles de processus trouvés dans les travaux de synthèse ne sont jamais des modèles correspondant à un système complexe de taille industrielle. La raison en est double, d'une part les automates synchronisés qui sont nécessaires à l'application de la théorie de la synthèse ne sont pas propices à la description de grands systèmes, et d'autre part l'explosion combinatoire inhérente à la théorie prend rapidement des proportions rédhibitoires pour les systèmes qui dépassent quelques entrées sorties. Il en va de même en vérification formelle, ou l'ajout d'un modèle de processus augmente la taille globale du modèle à vérifier et peut donc augmenter l'effet de l'explosion combinatoire. En revanche, les modèles de processus utilisés en simulation sont plus importants en taille, mais cela reste souvent le résultat d'un assemblage de modules qui sont inclus dans les logiciels du marché, plutôt que le résultat d'études scientifiques poussées, comme le montre [LEBRUN 2003]. Concernant les modèles de processus obtenus par identification on peut trouver traités des cas de taille industrielle comme par exemple dans [KLEIN 2005].

La nécessité de considérer les systèmes automatisés comme étant constitués de deux sous-systèmes (le contrôleur et le processus), principalement pour des contraintes d'ingénierie (développement parallèle des deux sous-systèmes), introduit une difficulté particulière. En effet le contrôleur et le processus évoluent de manière asynchrone avec quelques synchronisations. Pour l'identification sera posé le problème de détermination de l'instant de prélèvement des mesures (synchrone avec la perception qu'en a le contrôleur ou a l'instant de l'occurrence des événements). En synthèse, il y a le problème de concilier l'évolution des modèles à la suite de l'arrivée d'un événement non contrôlable, et enfin pour la vérification formelle il faut stipuler à quels instants et pour quels états doivent être faits les preuves de propriétés.

Dans tout les cas, l'utilisation d'un modèle processus n'a pas de sens si le modèle du processus utilisé contient des erreurs ou des imprécisions, on suppose par conséquent toujours que le modèle du processus est «crédible». Il est donc important de s'interroger sur la manière d'élaborer les modèles de processus

1.2 Comment obtenir le modèle du processus ?

Le modèle du processus peut être obtenu de diverses façons et en considérant différents formalismes. Basiquement, il existe deux approches pour la modélisation du processus : la modélisation monolithique et la modélisation modulaire.

1.2.1 Approche monolithique

La modélisation monolithique, comme nous la trouvons dans [RAUSCH & KROGH 1998] [HASSAPIS *et al.* 1998] [KOWALEWSKI & PREUßIG 1996] [MERTKE & MENZEL 2000] consiste en l'obtention de tout le modèle du processus en un seul bloc. Cela n'est ni modulaire, ni hiérarchique ni incrémental (par raffinement). Cette forme de modélisation se limite aux systèmes de petite taille. En effet, des modèles correspondant à des systèmes de taille industrielle ne sont tout simplement pas «réalisables» parce que le concepteur ne saurait pas les faire. Ils sortent des possibilités d'abstraction de l'esprit humain. En contrepartie de cette difficulté, dans les modèles monolithiques tous les états sont pertinents parce qu'ils ont été envisagés individuellement par le concepteur.

1.2.2 Approches modulaires

La modélisation modulaire consiste en à la «division» du processus en parties qui le constituent (modules) [ZAYTOON & CARRÉ-MÉNÉTRIER 1999] [BALEMI *et al.* 1993] [BARESI *et al.* 1998]. Un module est un élément du processus, comme par exemple une vanne, un vérin, un capteur, un moteur... Avec cette façon de modéliser, le point de départ consiste en l'élaboration du modèle de chaque module, mais le modèle final recherché est un modèle global et monolithique.

Ce modèle global est souvent obtenu au travers du produit cartésien des modèles de chaque module [ZAYTOON *et al.* 1999]. Dans ces conditions, le modèle obtenu a une dimension (en nombre d'états et transitions) supérieur à celui qui correspond au comportement réel du processus. Il y a des états et des transitions du modèle qui ne correspondent pas systématiquement à un comportement réel du système.

Une autre façon d'assembler des modèles modulaires est d'utiliser un formalisme permettant de décrire des modules qui communiquent entre eux [MACHADO *et al.* 2003-a], [MACHADO *et al.* 2003-b]. Néanmoins, dans ce type de modélisation le principal problème qui se pose est d'établir des règles de communication entre modules pour que le modèle global obtenu soit cohérent et traduise, réellement, le comportement du processus. Sans précaution, l'assemblage de modules peut également inclure des états et des transitions qui n'ont rien à voir avec le comportement du processus. Ce sont des états et des transitions caractérisant des situations transitoires de communication entre les modules.

De façon générale la modélisation modulaire permet une grande flexibilité vis à vis du besoin de l'utilisateur final :

- le modèle peut facilement se limiter à un sous ensemble du processus modélisé par assemblage d'une partie des modules,
- des modèles de différentes granularités peuvent être combinés,
- les modèles sont facilement appréhendables par les concepteurs,
- les modèles des modules sont plus facilement réutilisable, et peuvent donc être structurés en bibliothèque.

1.2.3 Bilan

Si la modélisation monolithique a l'avantage de présenter explicitement chaque étape et chaque évolution du comportement modélisé, son principal revers est d'être impraticable pour des processus de taille industrielle. Ainsi, la modélisation modulaire du processus est incontournable, mais l'obtention du modèle global au travers du produit cartésien des modèles correspondants à chaque module reste une opération délicate qui peu faire apparaître des états sans réalité physique.

1.3 Le cas de la vérification formelle par model-checking

C'est, sans aucun doute, dans le domaine de la vérification formelle que l'utilisation d'un modèle du processus est la moins explorée. L'objectif de cette section est de montrer l'intérêt et les difficultés d'utilisation d'un modèle de processus en vérification formelle par model-checking et de mettre en évidence quelques questions auxquelles il serait intéressant de répondre.

1.3.1 Exemple illustratif

Pour illustrer notre propos, un exemple académique très simple a été retenu. Il s'agit du système automatisé présenté figure 4 pour lequel le processus est composé de deux vérins pneumatiques à double effet, l'un horizontal et l'autre vertical, de deux distributeurs 5/2, l'un bistable (celui qui commande le mouvement horizontal, l'ordre «R» correspondant au mouvement vers la «droite» et l'ordre «L» correspondant au mouvement dans la direction «gauche») et l'autre monostable (qui commande le mouvement vertical, en étant l'ordre «D» correspondant au mouvement vers le «bas»). Mécaniquement, la tige du vérin horizontal et solidaire du corps du vérin vertical (liaison complète). Deux capteurs fin de course sont associés à chaque vérin, «l» et «r» étant associés au vérin horizontal pour indiquer les positions «gauche» et «droite», tandis que les capteurs «u» et «d» sont associés au vérin vertical pour indiquer les positions «haut» et «bas». La figure 4 présente également le SFC [IEC 1993] correspondant à la loi de commande du contrôleur.

L'objectif de ce système est de faire un cycle en «L», c'est-à-dire : sortie du vérin horizontal, descente du vérin vertical, montée du vérin vertical et rentrée du vérin horizontal.

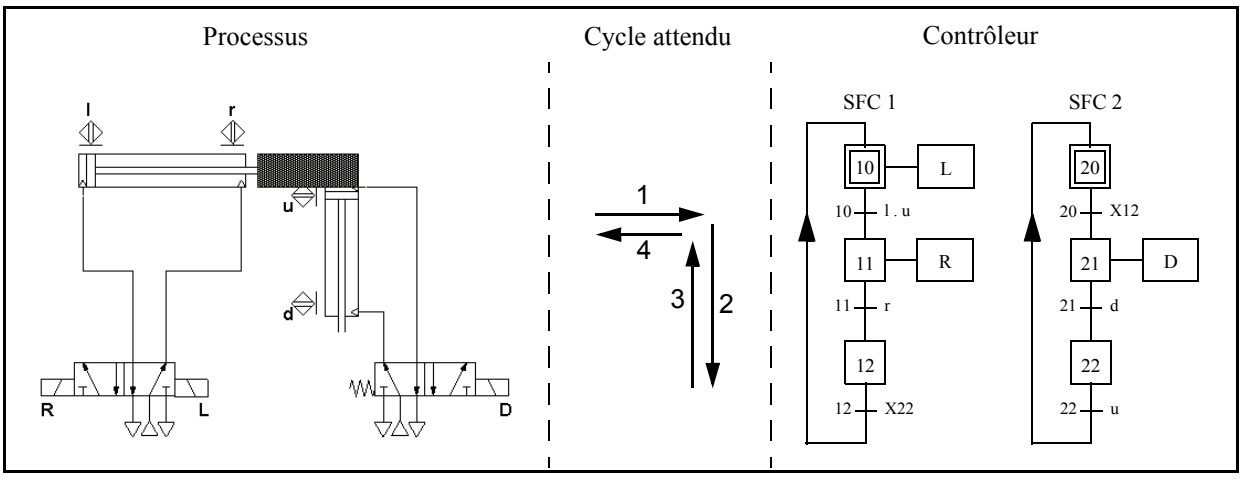


Figure 4: Exemple d'un système automatisé composé d'un contrôleur et d'un processus.

Le SFC de la figure 4, conçu pour implanter les spécifications du cahier des charges, comporte une erreur placée volontairement dans la transition 12 (la réceptivité aurait dû être «X22 . u» à la place de «X22»), ce qui a comme conséquence de rentrer les deux vérins au même instant, contrairement à ce qui était prévu dans le cahier des charges (la rentrée du vérin vertical doit se faire en premier avant la rentrée du vérin horizontal).

Les trois propriétés que l'on souhaite vérifier sont donc les suivantes :

- P1 : Les ordres R et L ne sont jamais émis simultanément,
- P2 : Lorsque le vérin horizontal reçoit un ordre de déplacement, alors le capteur «u» indique toujours que le vérin vertical est en haut,
- P3 : Il est impossible d'atteindre la situation pour laquelle le vérin horizontal est en position gauche et le vérin vertical est en position basse.

1.3.2 Preuve des propriétés

Pour un système de taille aussi réduite l'expert en analyse des SED est capable de conclure sur ces trois propriétés par un raisonnement simple.

Pour prouver que P1 est vraie, il suffit de regarder à quels moments les ordres «L» et «R» sont émis. «L» est émis quand l'étape 10 est active et «R» est émis quand l'étape 11 est active. Or le SFC2 étant un graphe d'état par construction (une seule étape active à la fois), les étapes 10 et 11 se sont jamais actives simultanément. La propriété P1 est donc vraie. Aucune hypothèse n'ayant été faite concernant le processus, on peut se demander quel serait l'intérêt de prendre en compte un modèle de processus pour cette preuve.

P2 est fausse, et cela peut se montrer à l'aide d'un scénario d'évolution ayant valeur de contre exemple. A partir de la situation initiale du contrôleur {10, 20} et de la position initiale du processus (les deux vérins rentrés avec donc l=1 et u=1), la transition 10 est franchie et la sortie «R» est émise ({11, 21} est la nouvelle situation du contrôleur). Le vérin horizontal sort, et lorsqu'il atteint sa fin de course, le capteur «r» transmet le signal logique «vrai» au contrôleur qui évolue en situation {12, 20} puis {12, 21}. Dans cette nouvelle situation le contrôleur émet la sortie «D», ce qui a pour effet la descente du vérin vertical. Lorsque ce dernier atteint sa fin de course, le capteur «d» transmet le signal logique «vrai» au contrôleur qui évolue en situation {12, 22}, puis {10, 22}. Dans cette situation où le vérin vertical est en position basse, le contrôleur émets la sortie «L». Il y a donc un ordre de déplacement pour le vérin horizontal alors que le capteur «u» n'indique pas que le vérin vertical est en haut. P2 est donc fausse. Cette fois ci, le comportement du processus à été pris en compte dans le raisonnement, et l'on peut se demander quel aurait été le résultat d'une preuve sans tenir compte du processus. Aurait-il été le même ?

Quant à la propriété P3, elle est vraie. En effet, l'étude des situations atteintes par le système composé du contrôleur couplé au processus (figure 5) met en évidence l'impossibilité d'atteindre la situation pour laquelle le vérin horizontal est en position gauche et le vérin vertical est en position basse. Cette fois encore, le comportement du processus a été pris en compte dans le raisonnement, mais aurait-il été possible d'exprimer cette propriété sans s'appuyer sur une représentation du processus ?

En résumé, les trois propriétés que l'on souhaite vérifier à l'aide de techniques de vérification formelle doivent nous conduire aux conclusions suivantes : P1 et P3 sont vraies, tandis que P2 est fausse. Mais faudra-t-il fournir ou non un modèle du processus au model-checker ? Si oui, dans quels cas de figure, c'est-à-dire pour quels types de propriétés ?

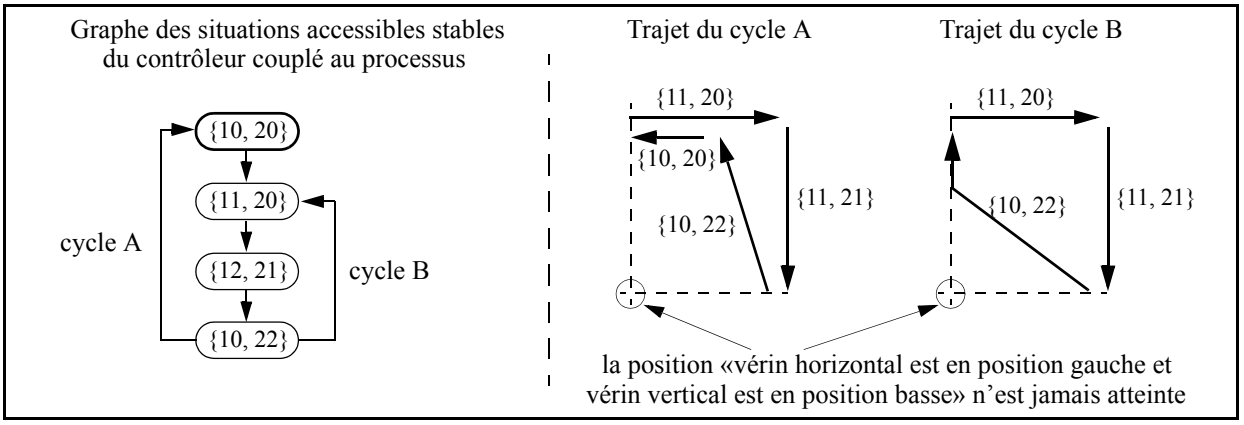


Figure 5: Etude des situations atteintes par le système (Contrôleur et Processus en boucle fermée)

1.3.3 Typologie et expression de propriétés

Les travaux de vérification formelle ont pour but principal la preuve de propriétés attendues du système étudié. La formulation des propriétés peut être faite à l'aide d'expressions en logique temporelle [CLARKE *et al.* 1986] [ALUR & HENZINGER 1991] ou, pour les propriétés plus complexes, avec un automate à états fini (dit automate observateur) associé à une expression simple en logique temporelle [DWYER & CLARKE 1994] [OLENDER & OSTERWEIL 1990].

On peut distinguer deux grandes familles de propriétés : les propriétés de sûreté et les propriétés de vivacité [ALPERN & SCHNEIDER 1985] [CORBETT & AVRUNIN 1995] [MANNA & PNUELI 1995] [CHEUNG & KRAMER 1999] [VAREA *et al.* 2002]. Les propriétés de sûreté sont des conditions qui doivent être vérifiées le long de toutes les évolutions du système et elles s'énoncent par exemple comme suit : «quelque chose se produit toujours» (ou sa contraposée, «quelque chose ne se produit jamais». Ces propriétés sont, normalement, associées à des comportements critiques du système. Les propriétés de vivacité décrive la possibilité qu'a le modèle d'évolution dans un état pour lequel des conditions qui sont vérifiées (existence de chemins). Elles s'énoncent par exemple comme suit : «il est possible d'atteindre une situation donnée», ou «il est toujours possible d'atteindre une situation donnée». Pour notre exemple (section 1.3.1 p. 13) les propriétés P1 et P2 sont des propriétés de sûreté, tandis que P3 est une propriété de vivacité.

Notre capacité à exprimer une propriété ainsi que notre capacité à conclure sur l'absence d'erreur dans le programme du contrôleur dépendent des éléments du système sur lesquels portent les propriétés. Nous distinguerons trois familles de variables utilisée dans l'expression des propriétés :

- les «variables contrôlables» par le contrôleur qui sont ses variables de sorties et ses variables internes, comme les états,
- les «variables observables» par le contrôleur qui sont ses variables d'entrée, et
- les «variables processus» qui décrivent la situation du processus à l'exclusion des informations des capteurs (variables observables) et des ordres aux pré-actionneurs (sous ensemble des variables contrôlables).

Les travaux de vérification formelle des contrôleurs utilisant une approche «non model-based» [DE SMET & ROSSI 2002] [BORNOT *et al.* 2000] ne peuvent exprimer que des propriétés à base de variables contrôlables et observables. En revanche l'approche «model-based» [MACHADO *et al.* 2003-a] [MACHADO *et al.* 2003-b] permet l'expression de propriétés incluant les trois familles de variables.

Pour notre exemple illustratif de la figure 4, la propriété P1 est formulée uniquement avec des variables contrôlables (les ordres «L» et «R»), la propriété P2 utilise à la fois des variables contrôlables (ordres de déplacement) et des variables observables (capteur «u»), et la propriété P3 s'appuie sur des variables processus (situations des vérins)

1.3.4 Granularité du modèle du processus

La seconde question qu'il est légitime de se poser à propos du modèle de processus est : quel est le niveau de détail (granularité) requis pour modéliser le processus ? Considérons, par exemple, le sous-ensemble fonctionnel électro-pneumatique (électro-vanne, vérin et capteurs) correspondant au vérin horizontal de notre exemple illustratif présenté figure 4. La figure 6 présente 3 niveau possible de granularité :

- un modèle à 2 états où seules les deux positions extrêmes sont prises en considération,
- un modèle à 3 états où en plus des deux positions extrêmes du vérin, on représente la situation intermédiaire,
- un modèle à 4 états où en plus des deux positions extrêmes du vérin, on distingue par deux états distincts le mouvement de sortie et le mouvement de rentrée de la tige.

Certains auteurs proposent également des modèles à 5 états [GOUYON 2001] voire même à 9 états [CARRÉ-MÉNÉTRIER *et al.* 2002]. Le choix du niveau de détail des modèles a-t-il une incidence sur la vérification ?

1.3.5 Formalisme à utiliser

Dans les travaux identifiés en vérification formelle avec utilisation d'un modèle du processus, comme par exemple [RAUSCH & KROGH 1998], [HASSAPIS *et al.* 1998], [KOWALEWSKI & PREUßIG 1996], et [MERTKE & MENZEL 2000], le processus est toujours modélisé avec une classe de formalisme à états : automates à états, systèmes condition/événement ou réseaux de Petri. Cependant le choix du formalisme retenu est généralement guidé par l'usage qui en sera fait plutôt que par des besoins liés à une modélisation modulaire d'un processus. Ainsi, indépendamment de l'usage auquel il sera destiné, un formalisme de modélisation des processus devrait avoir les caractéristiques principales suivantes :

- le non déterminisme, pour prendre en compte l'indéterminisme du processus commandé,
- une grande capacité d'expression pour prendre en compte la complexité des processus,
- une aptitude à la modularité.

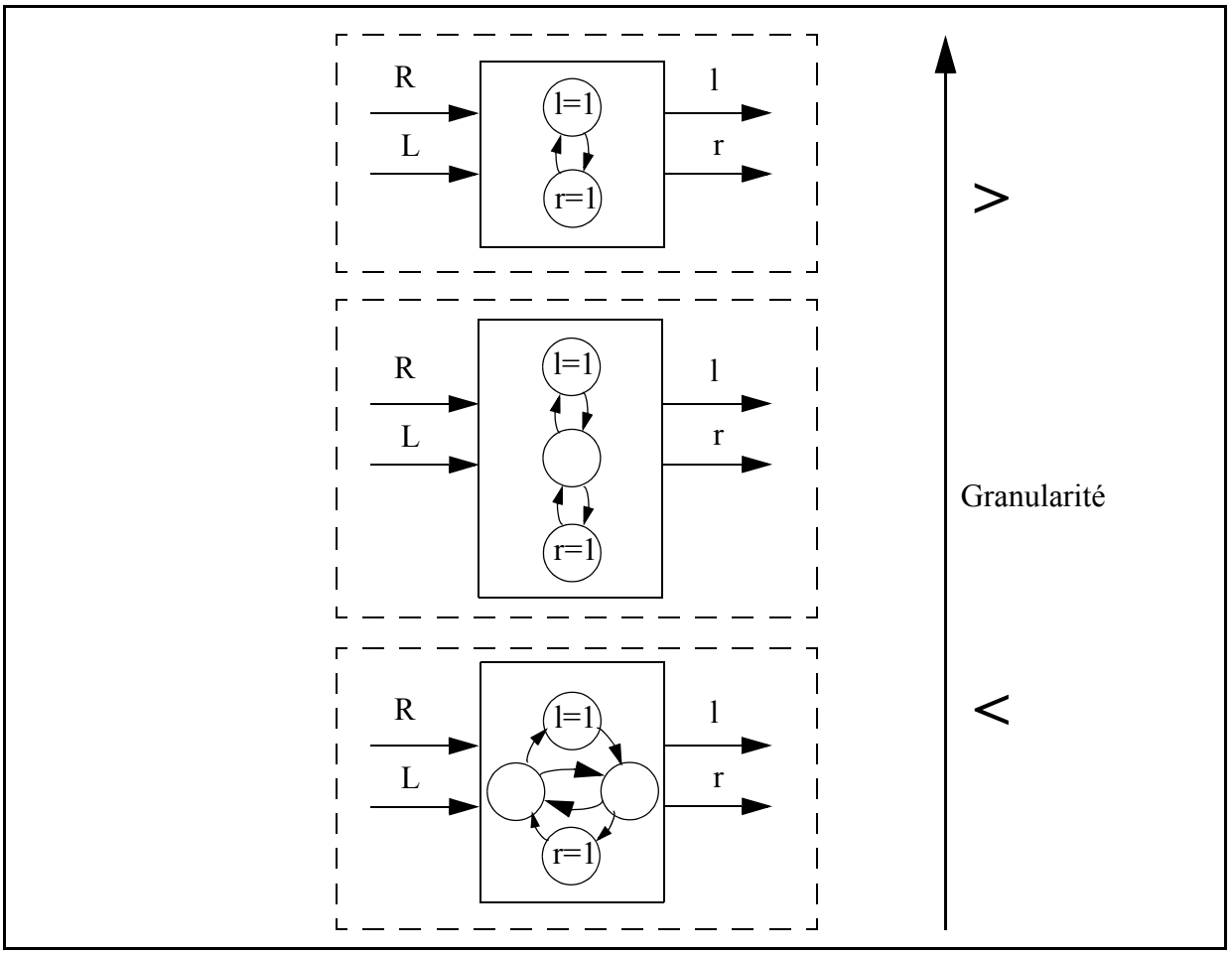


Figure 6: Différentes granularités pour un modèle d'un vérin pneumatique avec sont électro-vanne et deux capteurs fin de course

1.3.6 Couplage des modèles du contrôleur et du processus

Les évolutions dans le temps du contrôleur et du processus sont asynchrones. L'un évolue en parallèle de l'autre mais avec des instants de synchronisation. De plus, leur constante de temps correspond à deux échelles différentes, l'un est très réactif (de l'ordre de la milliseconde pour le contrôleur) tandis que l'autre est lent (de l'ordre du dixième de seconde pour le processus).

Ainsi l'exécution cyclique du contrôleur impose des resynchronisations ponctuelles avec le processus dans la phase d'émission de ses sorties et la phase de lecture de ses entrées. Toute variation des entrées du contrôleur imposée de manière asynchrone par le processus ne sera prise en compte par le contrôleur que dans sa prochaine phase de lecture. De manière non symétrique, le processus prend immédiatement en compte toute variation de ses entrées (sorties du contrôleur). Cependant il est généralement composé de plusieurs sous-processus qui procurent une réponse plus ou moins différée aux ordres reçus. La durée de réaction peut ainsi être bien supérieure ou inférieure à un temps de cycle du contrôleur.

La figure 7 illustre cette complexité dans la relation entre le contrôleur et le processus. Certains travaux ont déjà partiellement pris en compte cet aspect en intégrant l'exécution cyclique du contrôleur dans la vérification formelle [LAMPÉRIÈRE-COUFFIN & LESAGE 2000]. Mais l'approche retenue était «non model-based» et l'intégration d'un modèle de processus n'était pas

envisagée. La vérification formelle du comportement d'un contrôleur avec une approche de type «model-based» devra donc modéliser ce couplage des deux comportements contrôleur/processus.

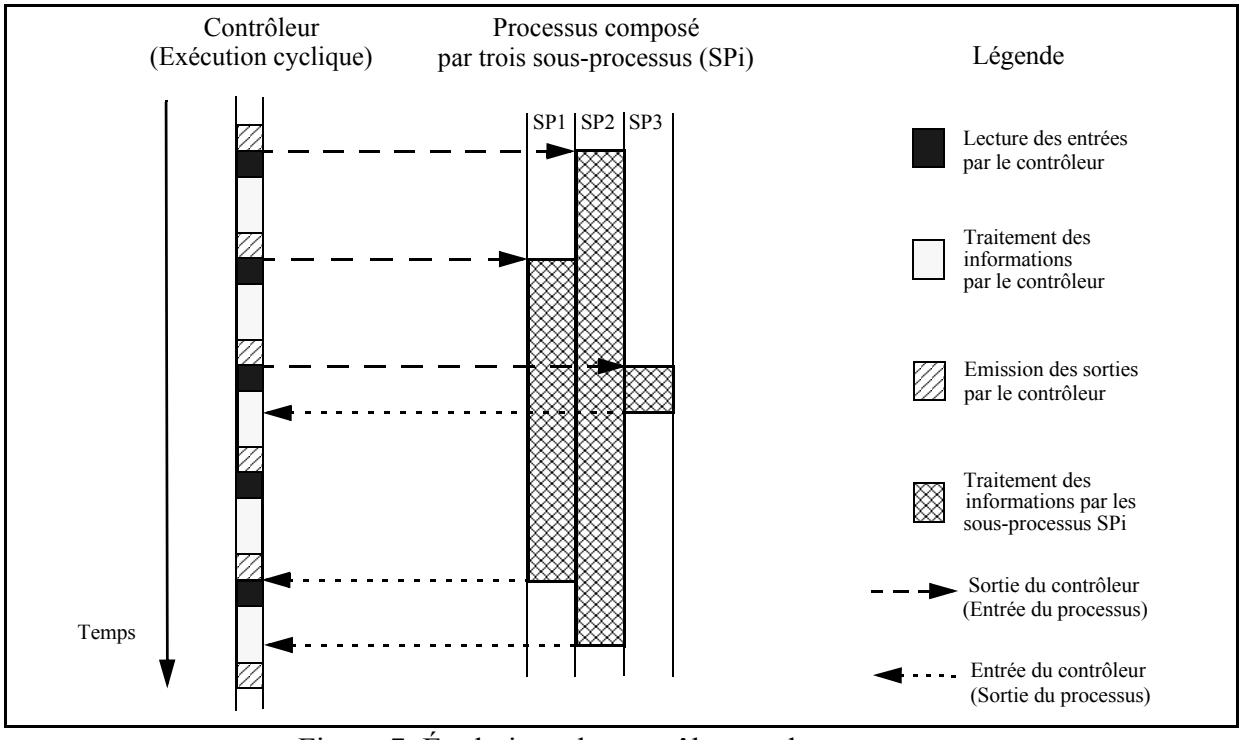


Figure 7: Évolutions du contrôleur et du processus

1.3.7 Bilan sur la vérification par Model-Checking

L'examen d'un exemple nous a donné un éclairage sur le rôle que peut jouer le comportement du processus en vérification des systèmes de production. Ce rôle est plus complexe que l'on pourrait l'imaginer au premier abord. En effet, dans certains cas il n'est pas nécessaire et ne semble jouer aucun rôle, tandis que dans d'autres cas il est indispensable à l'expression d'une propriété. Il est donc nécessaire de mieux cerner l'impact de l'utilisation d'un modèle de processus sur la qualité du résultat des preuves en vérification des contrôleurs par Model-checking, de manière à améliorer de la sûreté de fonctionnement des systèmes de production.

Par ailleurs, notre étude bibliographique a fait émerger quatre aspects qui peuvent influer sur l'efficience de la vérification :

- la nature des propriétés à vérifier, qui pour certaines nécessitent un modèle de processus, alors que pour d'autres ce n'est pas nécessaire,
- le niveau de détail du modèle de processus qui influe sur la pertinence des résultats et sur le temps de calcul nécessaire à la vérification,
- le pouvoir d'expression du formalisme de modélisation comportementale utilisé et son adéquation avec la modélisation des processus étudiés, et
- la logique de couplage des modèles du contrôleur et du processus.

1.4 Bilan du chapitre 1

Dans ce chapitre nous avons mis en évidence que les modèles de processus sont utilisés à des fins diverses dans le domaine des SED. Malgré cette diversité, deux points communs peuvent être

soulignés : les techniques de modélisation utilisées sont généralement de la famille des automates à états finis, et une description modulaire s'impose très vite dès que la taille du processus à modéliser est significative. Cependant, dès que la taille du modèle du processus augmente, et en particulier celle de son espace d'états atteignables, la praticabilité des techniques utilisées s'amenuise. C'est le cas de la synthèse de contrôleur par la théorie de la supervision, ou de la vérification par Model-Cheching. Il est donc important de produire des modèles efficaces au sens de leur compacité, c'est-à-dire avec un espace d'états atteignables le plus réduit possible pour modéliser un comportement donné.

Dans les travaux que nous avons étudiés, un aspect qui est souvent éludé est la nécessité de s'appuyer sur un modèle de processus "juste", de manière à produire des résultats qui ont un sens. C'est vrai en particulier pour la vérification formelle par Model-Cheching qui nous intéresse plus spécifiquement. Pour augmenter la confiance que nous pourrons accorder aux résultats de vérification, une technique rationelle et systématique de construction du modèle de processus sera proposée dans le chapitre 3. Mais au préalable, le chapitre suivant va être consacré à la présentation des hypothèses retenues et des choix effectués pour la suite de nos travaux.

Chapitre 2
Approche globale, hypothèses et choix retenus

Après avoir présenté les utilisations possibles d'un modèle du processus dans la conception des SED on montrera, dans ce chapitre, les directions adoptées pour l'élaboration de notre travail, ainsi que les hypothèses retenues pour le réaliser. Ces hypothèses concernent le contrôleur, le model-checker, l'expression des propriétés et les interactions entre le contrôleur et le processus.

Finalement on présentera le formalisme de modélisation retenu, basé sur une classe d'automates communicants par partage de variables. Ce formalisme nous permet de répondre à l'ensemble de nos besoins de modélisation, tant pour le contrôleur et le processus que pour l'expression de propriétés complexes.

2.1 Approche globale retenue

Afin d'améliorer la sûreté de fonctionnement des systèmes automatisés de production, le principal objectif de ce travail est de repousser les limites actuelles de la vérification formelle des programmes de contrôleurs industriels par model-checking, tout en augmentant le nombre de propriétés qu'il est possible de prouver grâce à l'utilisation d'un modèle du processus commandé.

Jusqu'à présent, afin de repousser les limites de la vérification formelle des programmes de contrôleurs industriels différentes apports peuvent être trouvés dans la littérature : prenant en compte les programmes des contrôleurs élaborés avec plusieurs langages [DE SMET *et al.* 2000]; utilisant des contrôleurs multi-tâches et en considérant le temps physique [GOURCUFF 2004]; prenant en compte un modèle du processus basique [RAUSCH & KROGH 1998], ...

Dans notre approche, nous proposons de repousser les limites de la vérification formelle des programmes des contrôleurs industriels par model-checking grâce à l'utilisation de modèles détaillés et réalistes du processus. Pour cela, nous considérerons un contrôleur industriel à exécution cyclique et monotâche, dont le programme est élaboré dans un seul langage de programmation et nous retiendrons une abstraction logique du temps (prise en compte d'une relation d'ordre strict entre les dates d'occurrences d'événements sans en quantifier la durée les séparant). Notre objectif est d'étudier l'influence d'un modèle du processus sur la qualité des résultats obtenus en vérification formelle.

Nous allons montrer qu'en model-checking, certaines propriétés sont plus pertinentes si elles sont vérifiés sans l'utilisation du modèle du processus (propriétés de sûreté) et que autres doivent être vérifiés avec l'utilisation du modèle du processus (propriétés de vivacité).

En effet, quand un modèle du processus n'est pas utilisé, toutes les évolutions du système sont envisagées, figure 8 (a), parce que le contrôleur n'a pas d'information sur le comportement réel du processus. Dans ce cas, la preuve de propriétés de sûreté (du type «pour tous les chemins, dans tous les états ...») est plus forte. En effet, si toutes les évolutions du processus sont envisagées, même celles qui n'arriveront jamais, cette propriété sera toujours vérifiée quand on prend en compte un modèle du processus.

Par contre, si le modèle du processus est pris en compte, figure 8 (b), seuls les comportements réalistes du système de commande sont considérés. Dans ce cas, les propriétés de sûreté, ne seront vérifiées que pour un sous-ensemble de chemins et d'états du comportement du contrôleur seul (figure 8 (b)). Dans ce cas, la preuve de la propriété de sûreté est moins forte, parce qu'elle est faite seulement pour un sous-ensemble de chemins et d'états de l'ensemble précédemment considéré. On montrera dans la suite du mémoire que la prise en compte d'un modèle du processus ajoute des états au modèle global vérifié, mais réduit le nombre d'états atteignables par le contrôleur.

Par ailleurs, si nous considérons une propriété de vivacité (du type «il existe un chemin, au long duquel tous les états ...») qui est prouvée en considérant le modèle du processus (moins de chemins d'évolution possibles et moins d'états atteignables), alors celle ci reste vérifiée sans prise en compte d'un modèle du processus. La preuve d'une propriété de vivacité sera donc plus forte avec un modèle de processus car dans ces conditions elle est prouvée en prenant en compte toutes les restrictions de comportement du contrôleur imposées par le comportement du processus.

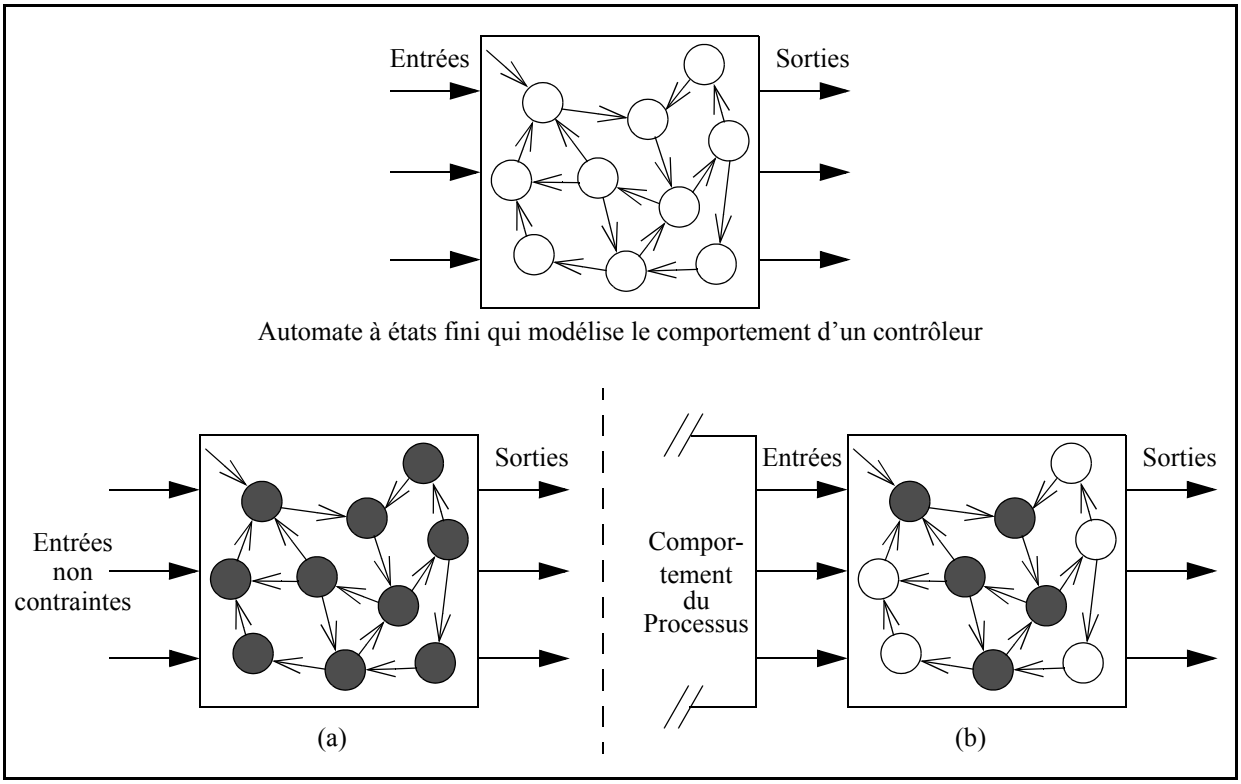


Figure 8: Influence d'un modèle de comportement du processus sur l'espace des états atteignables par le modèle du contrôleur.

Nous allons maintenant proposer :

- une méthode systématique pour construire et vérifier un système par une approche Model-Based ;
- une méthode systématique pour traduire un modèle de comportement dans l'outil de vérification formelle
- le traitement d'un exemple représentatif d'une classe de systèmes automatisés ;
- comparer les deux approches de vérification formelle Non Model-Based et Model-Based pour une base très variée de propriétés de sûreté et de vivacité ;
- dégager des règles de bonne utilisation des deux approches (Non Model-Based et Model-Based) pour en tirer le meilleur parti.

Pour ce faire, nous aurons besoin d'adopter quelques hypothèses de travail qui sont indiquées par la suite.

2.2 Hypothèses retenues

Nous allons retenir les hypothèses générales suivantes (illustrées sur la figure 9) :

- nous ne nous intéressons qu'aux systèmes à événements discrets logiques ;
- le temps est pris en compte d'une façon logique et non explicite ;
- l'ensemble du processus est commandé par un seul contrôleur ;
- le contrôleur a un comportement du type monotâche cyclique, et son programme est écrit dans un seul langage de programmation : SFC [IEC 1993] ;
- la vérification formelle sera faite par symbolic model-checking.

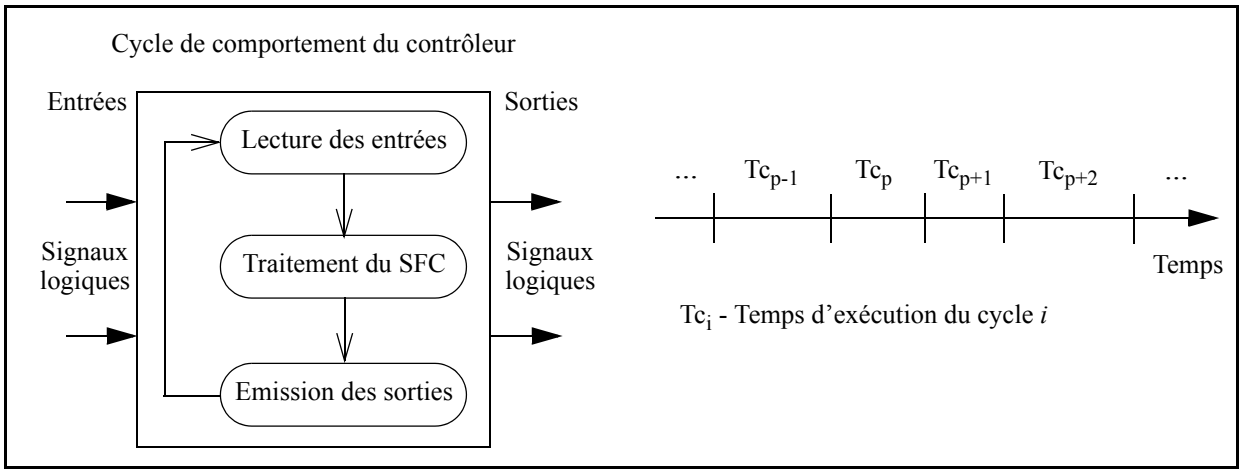


Figure 9: Hypothèses de travail pour le contrôleur.

2.2.1 Hypothèses liées aux propriétés

Les propriétés à vérifier peuvent être décrites par des expressions en logique temporelle [CLARKE *et al.* 1986] ou en utilisant conjointement un automate à états fini, appelé automate observateur, et des expressions en logique temporelle [DWYER & CLARKE 1994] [OLENDER & OSTERWEIL 1990] [CHEUNG & KRAMER 1999].

Dans notre travail nous avons retenu les logiques temporelles, CTL et LTL, dont on peut trouver une définition formelle dans [SCHNOEBELEN *et al.* 1999]. Ces deux logiques sont en effet bien adaptées à l'expression des propriétés qui ne nécessitent qu'une description logique du temps. Elles ont de plus des pouvoirs d'expression complémentaires : CTL spécifie des états tandis que LTL spécifie des chemins [EMERSEON & ALPERN 1986].

Lorsque l'utilisation d'un automate observateur est nécessaire, en complément de l'expression en logique temporelle, pour exprimer une propriété, nous utiliserons la même classe d'automates que celle retenue pour la modélisation du contrôleur et du processus (cf. 2.3, page 27).

2.2.2 Hypothèses concernant le model-checking

Nous avons retenu une approche de vérification de type «symbolic model-checking» parce qu'il s'agit d'une méthode éprouvée, outillée, et souvent plus simple à mettre en oeuvre que le theorem proving. En outre, l'expérience précédemment acquise au sein du LURPA dans ce domaine a conforté ce choix.

Le model-checking permet de répondre à la question suivante [CLARKE et al. 1999] : soit A le modèle du système à vérifier (exprimé par un automate) et φ l'expression formelle d'une propriété (exprimée en logique temporelle) : «Est-ce que le système A vérifie la propriété φ ?» (figure 10).

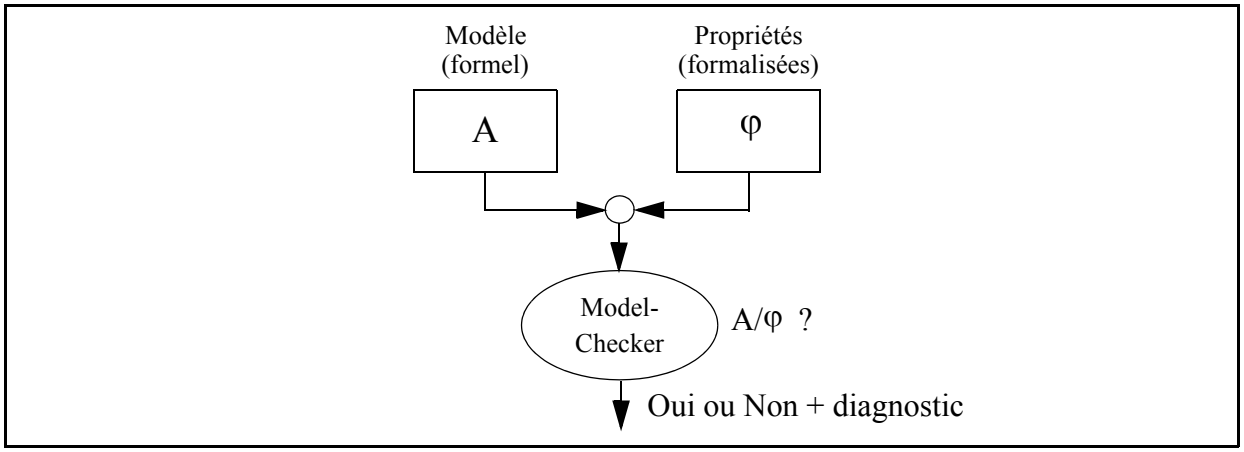


Figure 10: Problématique du model-checking

La principale limitation du model-checking est la taille de l'automate A . En effet, c'est la totalité de l'espace des états de A qui doit être parcourue pour déclarer la propriété vérifiée. C'est pourquoi, même dans les systèmes de taille réduite on peut assister rapidement à une augmentation rédhibitoire du nombre d'états atteignables du système, ce qui rend potentiellement la vérification formelle délicate pour les cas de taille industrielle.

Pour repousser les limites du problème de l'explosion combinatoire du nombre d'états atteignables on utilise une technique appelée *Symbolic Model-Checking* qui consiste en la non représentation explicite de tous les états atteignables en mémoire, mais dans sa représentation symbolique à travers la relation de transition du système. L'outil le plus souvent utilisé pour cela est le *OBDD (Ordered Binary Decision Diagram),* introduit par [BRYANT 1986], et plus tard appliqué au model-checking par [COUDERT *et al.* 1990], [BURCH *et al.* 1992], [PIXLEY 1992]. Pour nos expérimentations, nous avons retenu NuSMV [NuSMV 2002] eu égard à ces performances pour les SED logiques.

2.2.3 Hypothèses liées aux inter-actions contrôleur-processus

Dans le contrôleur, les ordres émis le sont toujours à la fin d'un cycle (émission des sorties). Leur prise en compte par le processus est supposée immédiate, mais leurs effets sur le processus ne seront détectables par le contrôleur que dans le cycle suivant, voire plusieurs cycles après. En effet, le temps de cycle du contrôleur est de l'ordre de quelques millisecondes tandis que le temps de réponse d'un composant du processus (ex: électro-vanne, vérin, ...) est de quelques centaines de millisecondes (figure 11). Dans un objectif de sûreté de fonctionnement, nous n'émettrons donc pas l'hypothèse simplificatrice que le contrôleur peut observer dans le cycle en cours les effets sur le processus des actions commandées dans le cycle précédent.

Comme illustré figure 11, pendant l'exécution d'un cycle du contrôleur il n'y a aucun échange d'information entre contrôleur et processus. Bien qu'évoluant en parallèle et de manière asynchrone, le processus et le contrôleur se «resynchronisent» à chaque cycle contrôleur (dans les phases de lecture des entrées et d'émission des sorties). Une transcription directe de ces comportements parallèles du processus et du contrôleur par deux automates évoluant en parallèle conduirait à l'exploration par le model-checker d'un très grand nombre d'états dûs à la composition de ces deux automates. Comme les propriétés que nous cherchons à prouver n'ont d'intérêt qu'aux instants de synchronisation entre contrôleur et processus, nous avons choisi de modéliser leur évolution par deux automates exécutés l'un après l'autre. Nous verrons dans le paragraphe 4.3, page 65, qu'une telle modélisation réduit la complexité de la vérification sans compromettre son intégrité.

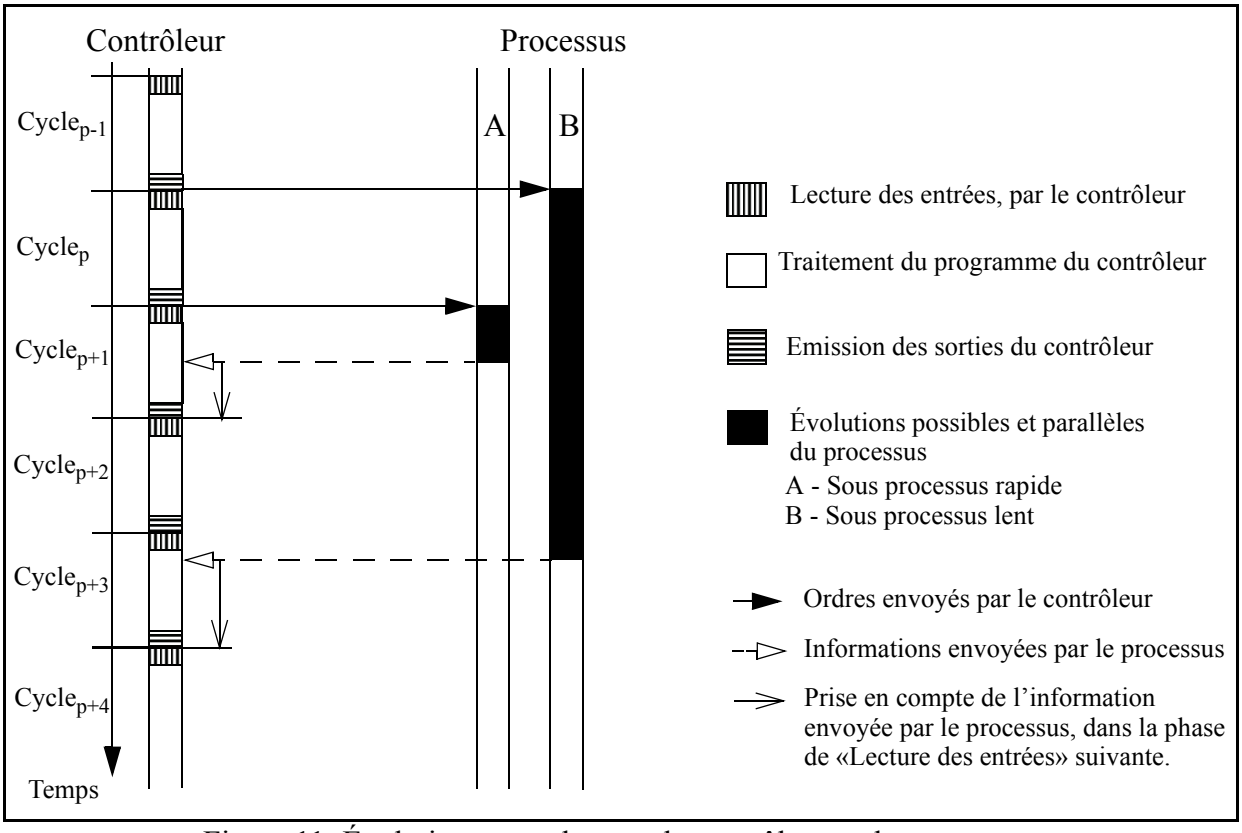


Figure 11: Évolutions asynchrones du contrôleur et du processus.

2.3 Formalisme de modélisation retenu

2.3.1 Caractéristiques attendues

Le formalisme retenu doit pouvoir être utilisé pour modéliser les comportements du processus commandé, du contrôleur et des propriétés exprimées sous la forme d'automate observateur.

Ce formalisme doit également être cohérent avec les hypothèses précédemment retenues :

- modélisation des systèmes logiques temporisés, avec prise en compte logique du temps ;
- capacité à utiliser une approche modulaire de construction des modèles ;

De plus, notre ambition étant de traiter des applications de taille industrielle, il nous faut rechercher un formalisme à fort pouvoir d'expression pouvant également exprimer du non déterminisme pour le comportement du processus [PHILIPPOT *et al.* 2004].

Une analyse bibliographique nous a conduit à retenir un formalisme de la famille des automates communicants. Les automates communicants [BRAND & ZAFIROPULO 1983] ont été introduits pour représenter des réseaux de processus communicants. Chaque processus est représenté par un automate à états fini qui communiquent par échange de messages. Plus tard, la communauté scientifique des systèmes hybrides a introduit le concept d'automates communicants par partage de variables [MALER *et al.* 1992] [ALUR *et al.* 1993] [ALUR & DILL 1994]. Dans ces automates le partage de variables est associé aux dynamiques continues, tandis que l'échange de messages reste utilisé pour les dynamiques discrètes. Cependant le principe même du partage de variable, qui procède par «diffusion» des valeurs dans les différents automates, favorise grandement la modularité dans la construction des modèles. C'est pourquoi nous avons défini une classe d'automates *communicant* par partage

de *variables logiques* et acceptant les expressions booléennes (au fort pouvoir d'expression) comme label de transition. Le choix des variables partagées plutôt que l'échange de messages est également motivé par la plus grande variété de synchronisations qui est offerte, ce qui va dans le sens d'un plus grand pouvoir d'expression.

2.3.2 Définition informelle de la classe d'automates retenue

L'automate que nous avons retenu est formé d'un ensemble de modules et d'un ensemble de variables logiques. Chaque module est un graphe orienté donc les noeuds sont appelés places et les arcs sont appelés transitions.

Les transitions sont labellées par une garde et un assignement. Les gardes sont des expressions booléennes sur les variables de l'automate, tandis que les assignements sont des listes d'affectation de variables.

Dans chaque module, une et une seule place est initiale. On appelle configuration de l'automate l'ensemble des place actives et des valeurs logiques de chaque variable à un instant donné.

Pour qu'une transition soit franchissable il faut à la fois que la place en amont soit active et que la garde associée soit vraie.

Si plusieurs transitions sont simultanément franchisables seule l'une d'entre elles est franchie. Elle est choisie au hazard.

2.3.3 Définition formelle de la classe d'automates retenue

Notre classe d'automate est basée sur la définition des automates hybrides définis par [ALUR & DILL 1994].

Définition 0 (Ensemble des expressions booléennes des variables du modèle) :

Soit V l'ensemble de toutes les variables booléennes du modèle :

L'ensemble $ExpBool_V$ des expressions booléennes des variables de V est défini par induction des assertions suivantes:

- quelque soit la variable $v \in V$, $v \in ExpBool_V$;
- $1 \in ExpBool_V$;
- $0 \in ExpBool_V$;
- Si $\tau \in ExpBool_V$ alors $\neg\tau \in ExpBool_V$
- Si $(\tau, \sigma) \in ExpBool_V^2$ alors $\tau \wedge \sigma \in ExpBool_V$;
- Si $(\tau, \sigma) \in ExpBool_V^2$ alors $\tau \vee \sigma \in ExpBool_V$.

Définition 1 (Automate) :

Un Automate global A est un triplé (M, V, e^0), où

- M est un tuple de modules $(m_1, ..., m_n)$, avec $n \geq 1$; on notera le $i^{éme}$ module de l'automate indifféremment m_i ou $M(i)$;
- V est un ensemble de variables booléennes ;
- e^0 est une fonction définie par $e^0 : V \rightarrow \{0, 1\}$, avec le domaine de e^0 égal à V.

Définition 2 (Module):

Un module *m* est un triplé *(P, T, p^0)*, où

- *P* est un ensemble non vide de places ;
- *T* est un ensemble non vide de transitions $p \xrightarrow{g, a} p'$ où :
  - $(p, p') \in P \times P$;
  - $g \in Gardes$ l'ensemble des gardes, où $Gardes \subset ExpBool_V$;
  - $a \in Assignements$ est un ensemble d' affectations : $a \subset V \times ExpBool_V$;
  - $p^0 \in P$ est la place initiale du module.

Définition 3 (Configuration) :

Une configuration *C* de l'automate est un couple (*S, e*) où :

- *S* est un tuple (p_1, ... p_n) de même taille que le tuple *M* avec $\forall i \in [1, n]$, $p_i \in M(i)$; *S* est la situation de *A* dans la configuration *C* et on appellera p_i la place active du module m_i dans la configuration *C* ;
- *e* est une fonction défini par :
  - $e : V \rightarrow \{0, 1\}$ avec domaine de *e* égal à *V*.

2.3.4 Règles d'évolution

On appelle transition franchissable d'une configuration *C* de l'automate *A* une transition $p \xrightarrow{g, a} p'$ telle que :

- $\exists i \in [1, n]$ tel que *S(i)=p* , et
- la valuation de l'expression booléenne *g* est égale à 1.

Une évolution de l'automate *A* depuis une configuration *C* est le résultat du franchissement d'une et une seule transition franchissable. La nouvelle configuration *C'* de l'automate est obtenue comme suit, par le franchissement de la transition franchissable $p \xrightarrow{g, a} p'$:

- La situation *S* devient la situation *S'* en replacent *p* par *p'* pour le module qui contient la place *p* ;
- Les assignements *a* sont appliqués aux variables concernées ;
  - Exemple: Si $a = \{(v1, \exp 1), (v2, \exp 2)\}$, la nouvelle valeur de la variable *v1* est le résultat de l'évaluation de l'expression *exp1* et la nouvelle valeur de la variable *v2* est le résultat de l'évaluation de l'expression *exp2*.

Remarque:

Les règles d'évolution ne précisent pas laquelle des transitions est franchie si plusieurs d'entre elles sont franchissables. Cela nous fourni le non déterminisme souhaité.

2.3.5 Notations et représentation graphique

Pour représenter graphiquement les automates, nous avons utilisé l'éditeur de l'outil UPPAAL [UPPAAL 2004]. Nous avons de plus profité des capacités de simulation de cet environnement pour la mise au point de nos modèles, avant de les vérifier formellement dans le model-checker NuSMV.

Conformément aux règles d'édition imposées par UPPAAL ;

- l'automate global a toujours un nom attribué
- chaque module a toujours un nom attribué ;
- les places sont représentés par des cercles ;
- la place initiale d'un module est représentée par un double cercle ;
- les transitions sont représentées par des arcs orientés entre deux places ;
- dans les expressions booléennes, on notera :
  - le «et» logique par «&&» ;
  - le «ou» logique par «||» ;
  - le «non» logique par «!» ;
- l'affectation d'une valeur à une variable dans un assignement est noté «:=».
  Par exemple :
  - V_P2 := 1, signifie que la valeur logique «1» a été assignée à la variable V_P2.

A titre d'exemple pour notre formalisme considérons l'automate *Aut* = (*Modules, Variables,* e^0) dont une représentation graphique est donnée figure 12, et qui est tel que :

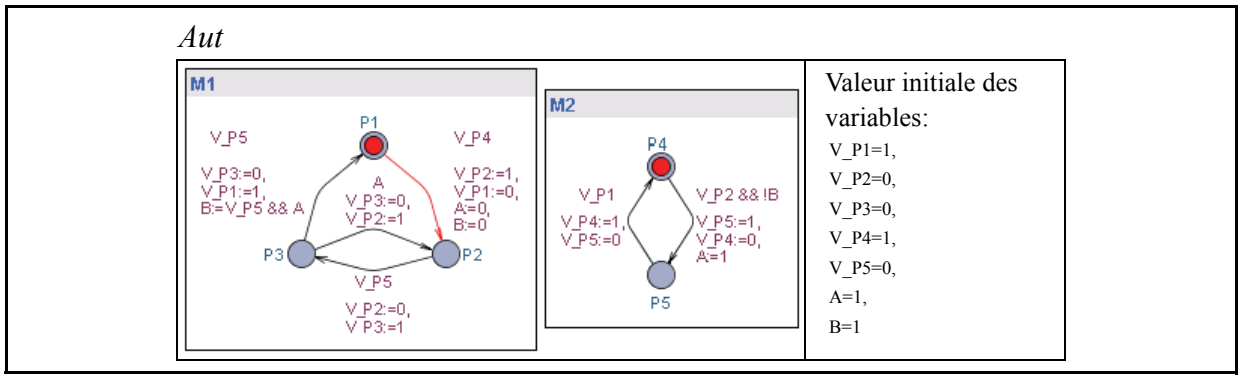


Figure 12: Représentation graphique de *Aut* (exemple illustratif de notre formalisme).

- *Modules* = (M1, M2)
  - M1 = (*Places*1, *Transitions*1, P1)
  - *Places*1 = {P1, P2, P3}
  - *Transitions*1 = $\{ \text{P1} \xrightarrow{\text{V\_P4, } \{(\text{V\_P2, 1}), (\text{V\_P1, 0}), (\text{A, 0}), (\text{B, 0})\}} \text{P2}$

    $\text{P2} \xrightarrow{\text{V\_P5, } \{(\text{V\_P2, 0}), (\text{V\_P3, 1})\}} \text{P3}$

    $\text{P3} \xrightarrow{\text{A, } \{(\text{V\_P3, 0}), (\text{V\_P2, 1})\}} \text{P2}$

    $\text{P3} \xrightarrow{\text{V\_P5, } \{(\text{V\_P3, 0}), (\text{V\_P1, 1}), (\text{B, V\_P5 \&\& A})\}} \text{P1} \}$
  - M2 = (*Places*2, *Transitions*2, P4)
  - *Places*2 = {P4, P5}
  - *Transitions*2 = $\{ \text{P4} \xrightarrow{\text{V\_P2 \&\& !B, } \{(\text{V\_P5, 1}), (\text{V\_P4, 0}), (\text{A, 1})\}} \text{P5}$

    $\text{P5} \xrightarrow{\text{V\_P1, } \{(\text{V\_P4, 1}), (\text{V\_P5, 0})\}} \text{P4} \}$
- *Variables* = {V_P1, V_P2, V_P3, V_P4, V_P5, A, B} et
- e^0 telle que :
  - $e^0 : V \rightarrow \{0, 1\}$ telle que :
  - e^0(V_P1) = 1
  - e^0(V_P2) = 0

- e^0(V_P3) = 0
- e^0(V_P4) = 1
- e^0(V_P5) = 0
- e^0(A) = 1
- e^0(B) = 1

Les évolutions de l'automate *Aut* à partir de la configuration initiale sont données dans le graphe des configurations accessibles de la figure 13.

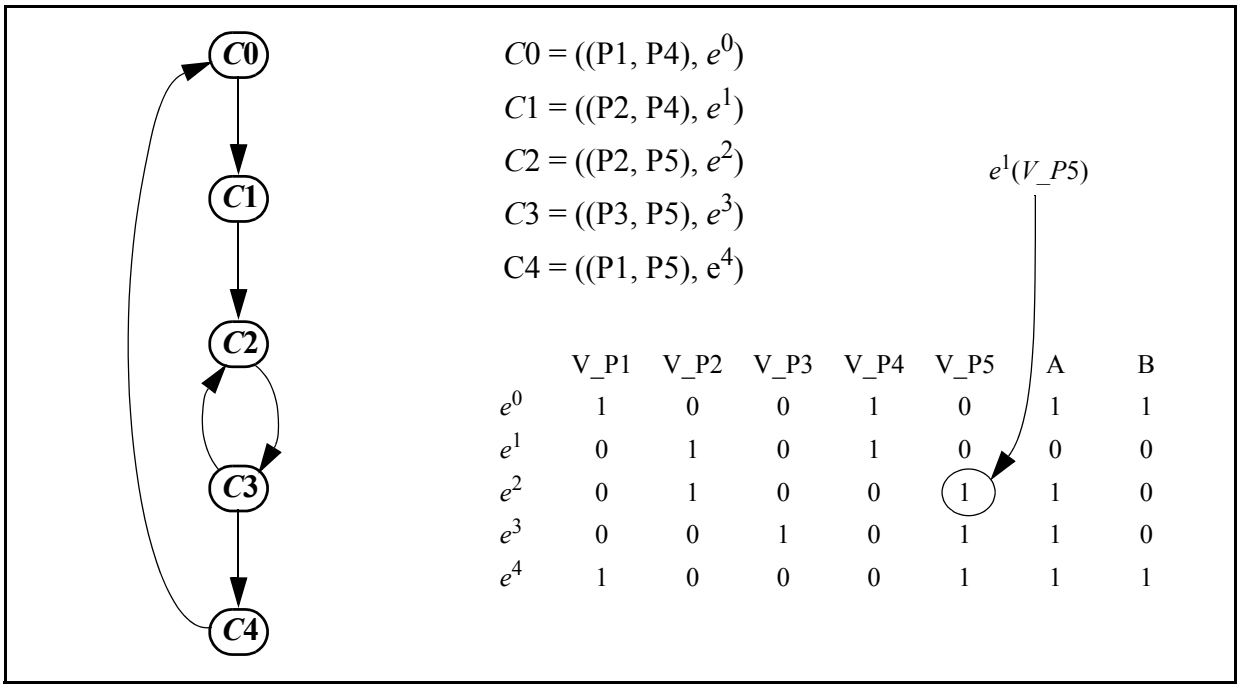


Figure 13: Configurations possibles pour l'évolution de *Aut*.

Le principe d'élaboration du graphe n'est pas détaillé dans ce mémoire. Il a été élaboré manuellement et uniquement pour illustrer les différentes configurations et évolutions de *Aut* au lecteur. La représentation graphique de chaque configuration est présentée dans la figure 14.

2.4 Bilan du chapitre 2

Ce chapitre nous a permis de présenter les hypothèses de travail retenues pour notre étude. Nous considérons les systèmes de production comme des systèmes à événements discrets logiques dans lesquels le temps est également pris en compte de manière logique. Ces SED sont composés d'un contrôleur et d'un processus qui interagissent en boucle fermée. Leurs comportements sont asynchrones, synchronisés lors des échanges d'informations (cadencés par les phases de lecture des entrées et d'affectation des sorties du contrôleur).

Pour la modélisation comportementale de ces SED, nous avons défini une classe d'automates spécifique au fort pouvoir d'expression. Il s'agit d'automates non déterministes, communiquant par partage de variables logiques et aux transitions labellées par des expressions booléennes. Cette classe d'automates est conçue pour permettre une modélisation modulaire du SED complet (contrôleur et processus). Le chapitre suivant est consacré à la proposition d'un cadre méthodologique rigoureux et systématique pour la construction du modèle du processus.

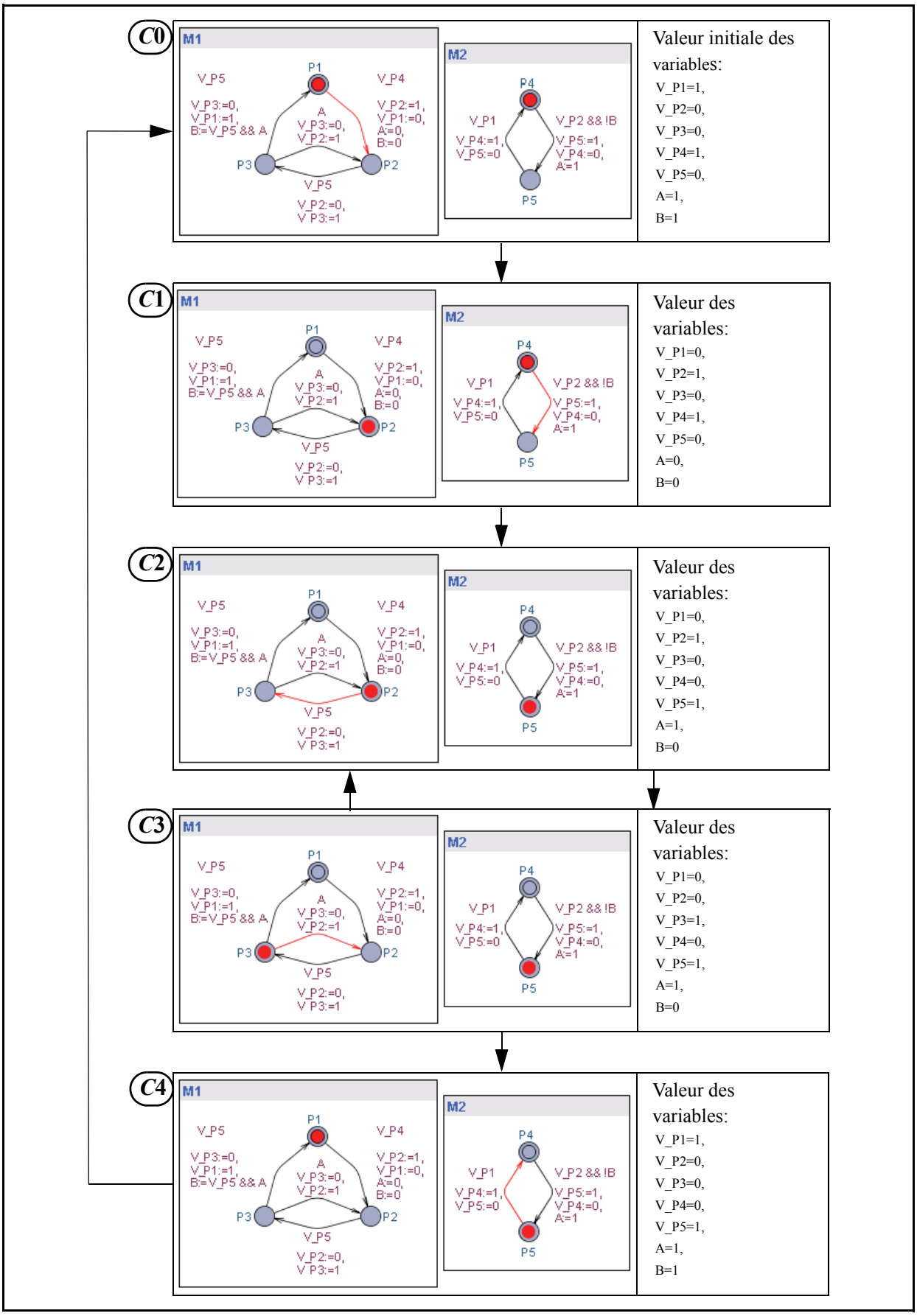


Figure 14: Exemple illustratif de notre formalisme; évolution du modèle.

Chapitre 3
Modélisation du processus

Dans ce chapitre on va montrer comment réaliser la modélisation du processus. Comme précédemment indiqué, notre approche est modulaire et privilégie la réutilisabilité, ce qui est indispensable à la modélisation des systèmes complexes. Nous présentons également un ensemble de règles qui, si elles sont appliquées systématiquement, peuvent rendre la tâche de modélisation du processus plus aisée grâce à un cadre méthodologique rigoureux.

Comme on l'a déjà mentionné dans les chapitres précédents, la modélisation du processus est essentielle pour la preuve de certaines propriétés comportementales des systèmes automatisés. Dans ce chapitre on abordera systématiquement toutes les phases de notre démarche de modélisation et on exposera l'ensemble des choix qui ont été réalisés comme, par exemple, la décomposition du processus en «modules» et la détermination de la granularité des modèles construits. Enfin, on rappelle que nous nous intéressons à la preuve des propriétés qui concernent les systèmes automatisés *en condition de bon fonctionnement*. Dans nos modèles de processus, les défaillances ne sont donc pas prises en compte.

3.1 Notion de modules

Deux aspects complémentaires caractérisent le niveau de granularité du modèle de processus que nous devons réaliser : le découpage du processus global en sous-ensembles et le niveau de détail du comportement modélisé pour chacun de ces sous-ensembles. Nous devons donc procéder à des choix qui concernent :

- les sous-ensembles opératifs (que nous appellerons «modules») que nous retenons pour la modélisation. Nous avons associé à ce choix la notion de «granularité de structuration» ;
- le niveau de détail avec lequel nous devons modéliser le comportement des modules. Nous avons associé à ce choix la notion de «granularité de comportement» ou de «finesse».

3.1.1 Granularité de structuration

La première question que le concepteur d'un modèle du processus se pose est : quels sont les différents modules que nous devons retenir pour la modélisation ? En effet, nous avons déjà indiqué, au travers d'une étude bibliographique dans le chapitre 1, que la modélisation monolithique était irréaliste pour les systèmes complexes. A l'inverse, l'association de modèles comportementaux à des sous-ensembles opératifs trop «atomiques» rejette le problème de modélisation du processus complet à l'assemblage de multiples micro-comportements. De notre point de vue, la granularité du découpage en modules doit donc être adaptée à l'utilisation qui en sera faite : dans notre cas elle doit permettre efficacement la preuve des propriétés de sûreté et de vivacité.

Considérons par exemple un système électro-pneumatique composé de plusieurs chaînes fonctionnelles dont certaines sont indépendantes et d'autres sont inter-connectées (figure 15).

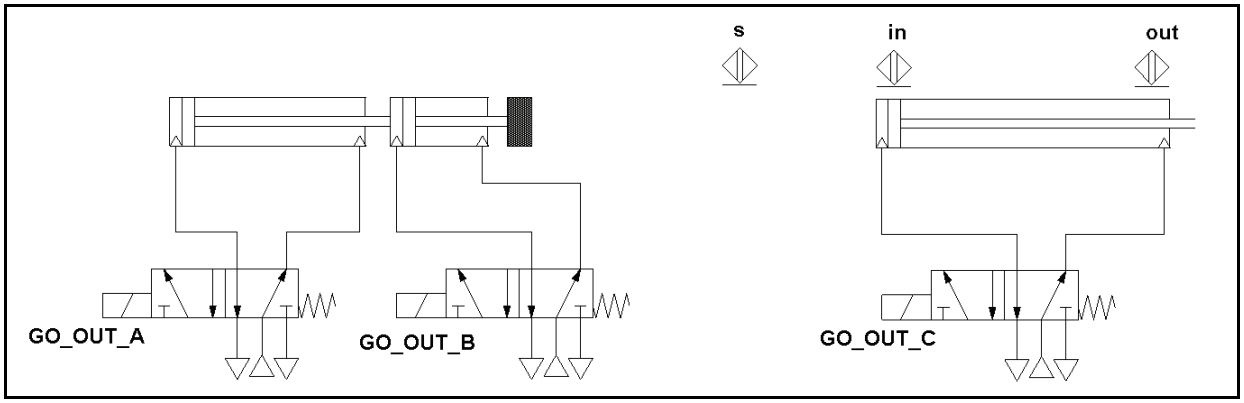


Figure 15: Exemple d'un processus électro-pneumatique

Une première approche consisterait à modéliser ce processus en un seul module, comme représenté au niveau 1 de la figure 16. Une autre possibilité serait de modéliser chacune des chaînes fonctionnel-

les qui le composent. Il est en effet aisé de trouver, pour chaque chaîne fonctionnelle, une relation directe entre les ordres envoyés par le contrôleur, le mouvement des solides et les états des capteurs (niveau 2, de la figure 16). Enfin, une autre possibilité serait de modéliser chacun des composants de chacune des chaînes fonctionnelles (solénoïdes des électro-vannes, vannes, vérins, capteurs,...) (niveau 3, de la figure 16).

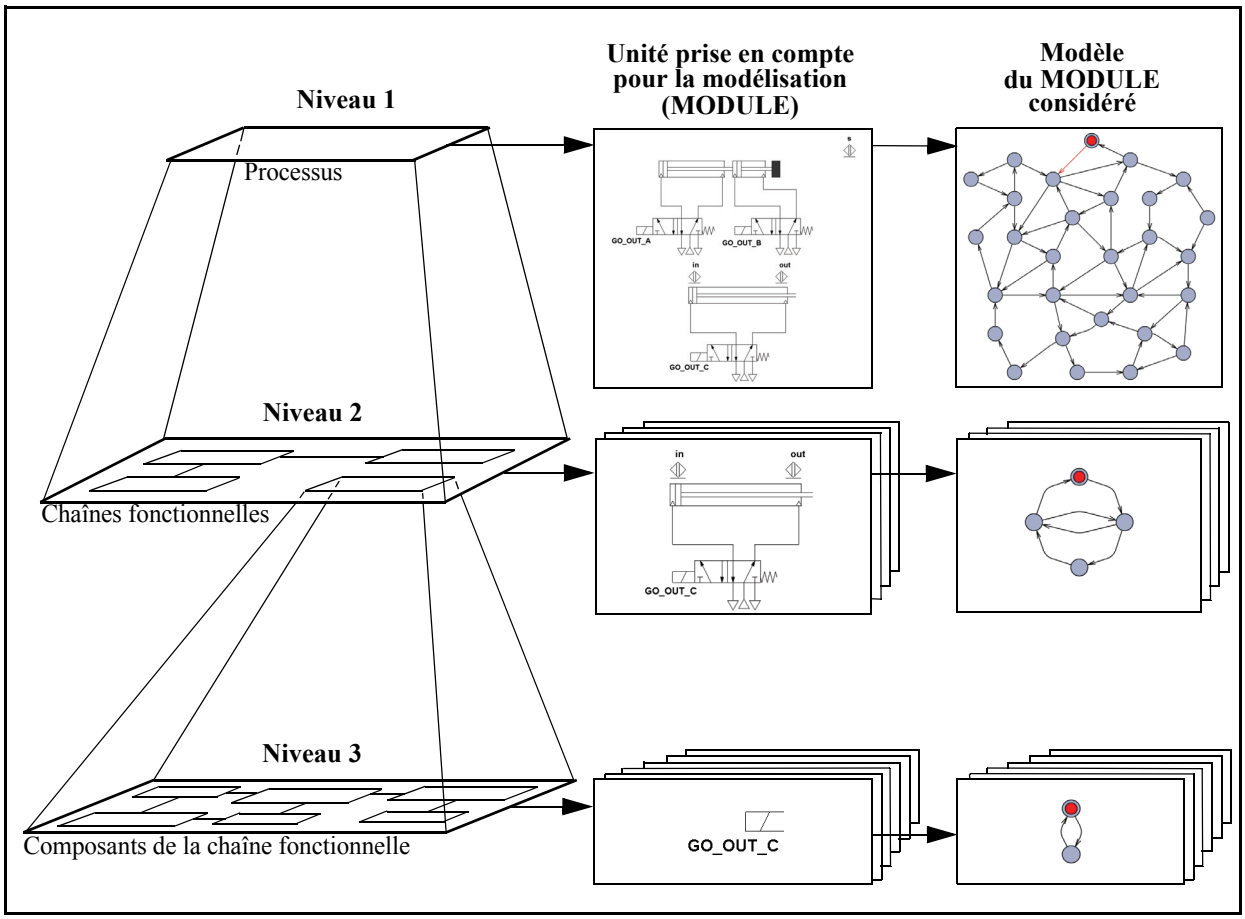


Figure 16: Granularité de structuration: différents niveaux possibles

Pour nos travaux, nous avons choisi le niveau de granularité de structuration correspondant au niveau 2 (au niveau de la chaîne fonctionnelle) pour les raisons suivantes :

- compte tenu des expérimentations que nous avons réalisées, la modélisation correspondant au niveau 1 (modélisation monolithique) est impraticable pour les systèmes complexes et ne permet aucune modularité ni réutilisabilité ;
- la taille des modules correspondant aux chaînes fonctionnelles (niveau 2) permet d'établir facilement une relation directe entre les entrées (*ordres* envoyés par le contrôleur), les *mouvements* des solides et les *sorties* (états des capteurs). Ces trois types d'informations sont utilisées dans l'expression des propriétés que nous voulons prouver. De plus, il est parfaitement possible, à l'échelle humaine, de modéliser des modules de ce niveau de complexité ;
- une modélisation correspondant au niveau 3 (composants élémentaires de la chaîne fonctionnelle) procure une modélisation plus fine mais le résultat obtenu en terme de preuve de comportement est exactement le même. Par contre, les modèles étant de plus grande taille l'effort de calcul nécessaire pour le model-checking est plus important. Il convient de noter que si on s'intéressait aux preuves de propriétés sur comportements fautifs, ce niveau de modélisation serait requis pour représenter les défaillances des composants du processus.

Dans la suite de nos travaux nous associerons donc un module à chaque chaîne fonctionnelle, sauf si certains composants du processus dépendent de plus d'une chaîne fonctionnelle. Dans ce cas, ces composants seront considérés eux-mêmes comme un module. Cet aspect sera développé dans la section "Modélisation de plusieurs chaînes fonctionnelles utilisant des composants communs", page 41

3.1.2 Granularité de comportement

Une autre difficulté majeure du concepteur des modèles du processus est le choix d'une granularité de comportement adaptée à l'objectif de son modèle. Le dilemme est bien sûr dans le cas du model-checking de trouver le bon compromis entre la taille du modèle (qui est porteuse de difficultés pour la vérification) et sa capacité à permettre la vérification des propriétés requises. Par exemple, si nous prétendons prouver une propriété qui prend en compte le mouvement de sortie de la tige d'un vérin nous aurons besoin d'un modèle où il apparaisse explicitement une grandeur (comme une place d'un automate) qui modélise ce mouvement de sortie. Nous allons maintenant indiquer quelle est la démarche de modélisation que nous avons retenue.

3.1.2.1 Modélisation de chaînes fonctionnelles

Pour illustrer notre démarche, nous allons utiliser l'exemple de la chaîne fonctionnelle présentée dans la figure 17. Plusieurs solutions s'offrent à nous pour la modélisation comportementale de ce module. On trouve du reste dans la littérature des modèles de cette même chaîne fonctionnelle qui comportent 2, 3, 4 (comme représenté sur la figure 18), 5 [GOUYON 2004] ou 8 places [ZAYTOON, CARRÉ-MÉNÉTRIER 1999] pour la chaîne fonctionnelle.

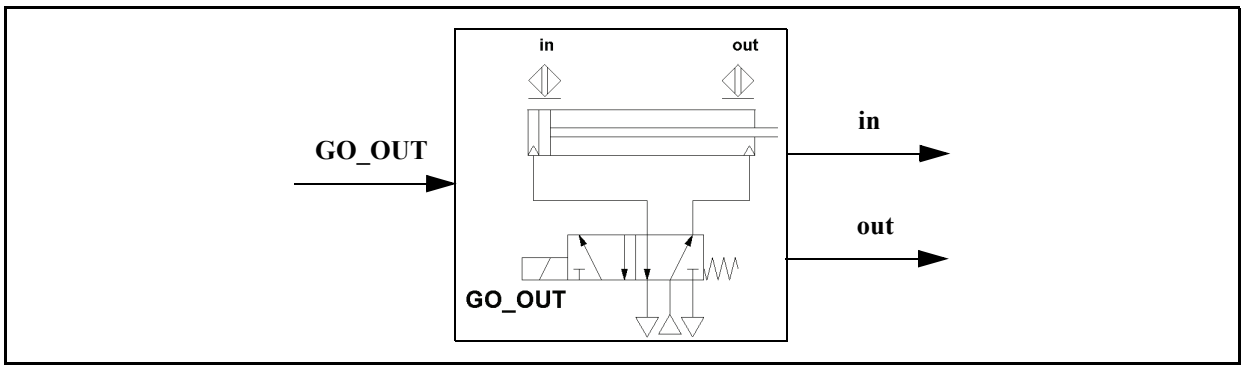


Figure 17: Chaîne fonctionnelle pneumatique servant d'exemple.

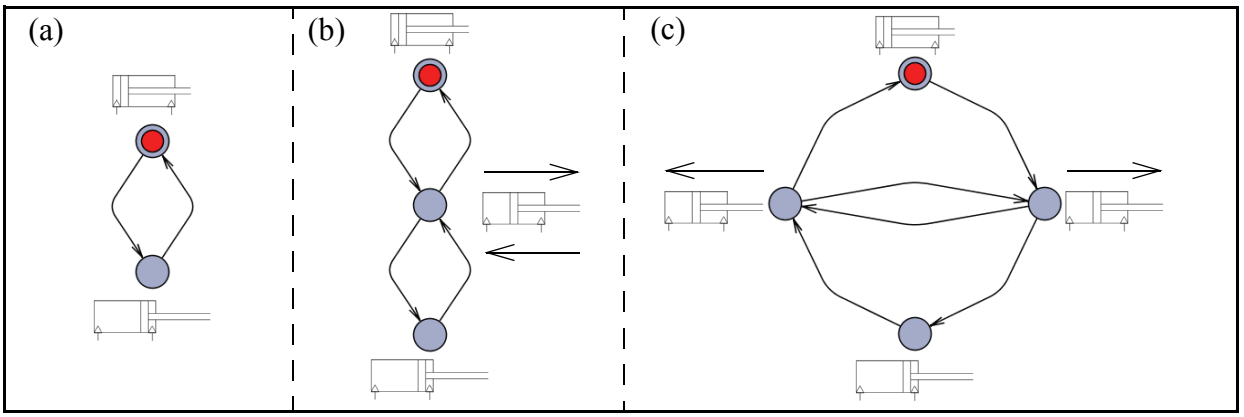


Figure 18: Quelques granularités de comportement pour le modèle d'une chaîne fonctionnelle de type «vérin double effet» : (a) deux places ; (b) trois places ; (c) quatre places.

Les règles suivantes ont été établies par expertise. Nous allons les retenir pour guider la construction de nos modèles :

- le comportement de chaque capteur doit être modélisé. Ce comportement peut être englobé dans le modèle global d'un module ou, lorsque ce capteur intervient dans plusieurs chaînes fonctionnelles, il doit faire l'objet d'un modèle particulier ;
- tous les mouvements de solides doivent être modélisés en distinguant explicitement tous les sens de chaque mouvement ;
- le modèle doit respecter les séquencements dûs à la réaction du processus aux ordres du contrôleur. Par exemple, dans la chaîne fonctionnelle pneumatique présentée plus haut, la propagation de l'ordre «*GO_OUT*», jusqu'à ce qu'il provoque le mouvement de la tige du vérin, est soumise à plusieurs retards (figure 19) :
  - retard pour que le signal électrique ait des effets dans la bobine (1, figure 19),
  - retard pour que le préactionneur change de position (2, figure 19),
  - retard pour que soit créée la pression suffisante pour commencer à bouger la tige du vérin (3, figure 19),
  - retard pour que la tige du vérin bouge et que le capteur «*in*» cesse de détecter le piston (4, figure 19).

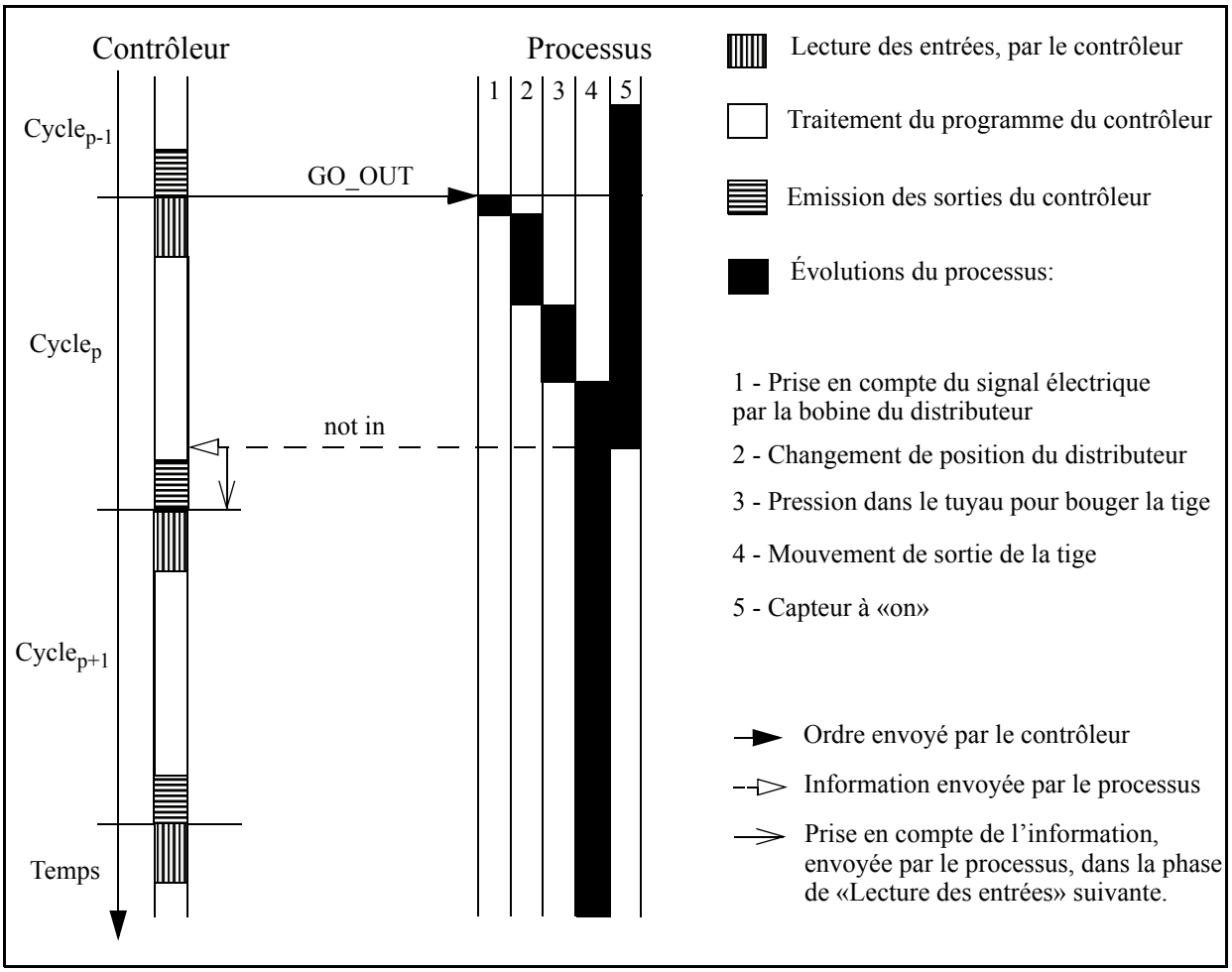


Figure 19: Décomposition du mouvement de la tige du vérin.

La première étape, pour la modélisation d'une chaîne fonctionnelle, consiste à fixer le nombre de places utilisées le modèle. Dans notre cas (figure 20), nous considérons quatre places qui décrivent le mouvement de la tige du vérin.

Parmi ces quatre places (*locations*, en terminologie anglaise), deux («*l_in*» et «*l_out*») caractérisent les positions extrêmes (rentré et sorti), deux autres représentent les configurations où la tige se trouve en mouvement entre ces deux positions extrêmes (en train de sortir et en train de rentrer : «*l_going_out*» et «*l_going_in*»). Les états des capteurs, «*in*» et «*out*» sont également associés à des places de ce modèle : «l_in» et «l_out», car ces deux capteurs ne sont utilisés que dans cette seule chaîne fonctionnelle. La place «l_in» est choisie comme place initiale, ce qui signifie qu'en l'absence d'ordre envoyé par le contrôleur, la tige du vérin est supposée rentrée.

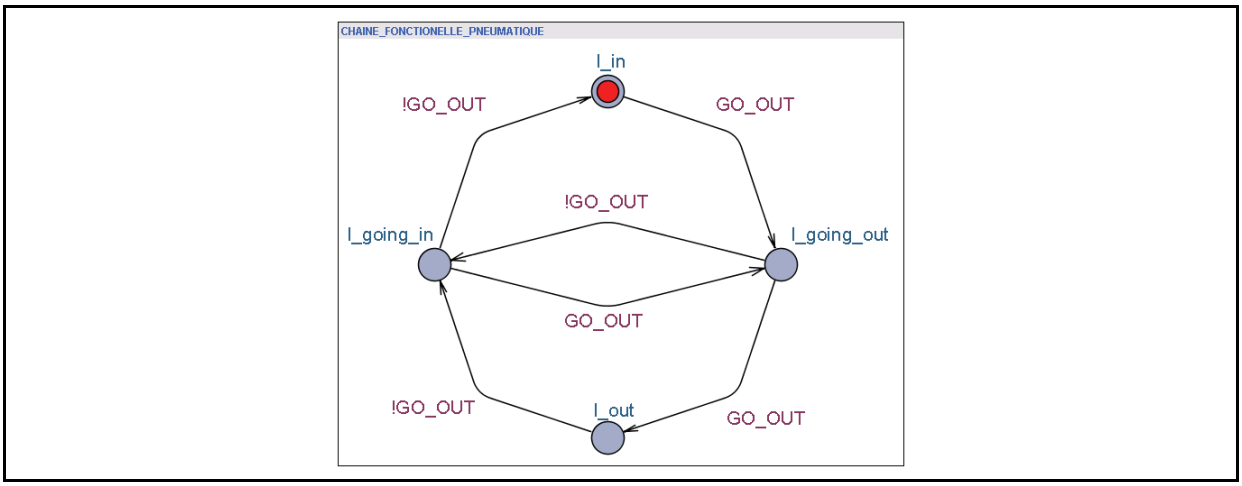


Figure 20: Modèle correspondant à la chaîne fonctionnelle pneumatique de la figure 17.

A chaque transition de l'automate est associée une garde, qui est une expression booléenne qui permet l'évolution du modèle entre les places. Ainsi, pour que le modèle évolue de la place «l_in» à la place «l_out» il faut que la variable «GO_OUT» ait la valeur logique «1». Une fois en «l_going_out» deux évolutions sont possibles : vers «l_out», qui représente la fin de course sortie ou vers «l_going_in» qui modélise le comportement si l'ordre «GO_OUT» cesse d'être émis avant l'arrivée de la tige à sa fin de course. La rentrée du vérin est obtenue de manière similaire.

3.1.2.2 Affectation des variables du modèle

Tous nos modèles de processus sont constitués par un tuple de modules et par un ensemble de variables dont la valeur logique change à chaque fois que le modèle évolue. Dans la configuration initiale du modèle toutes les variables ont une valeur logique initiale fixée. Concernant l'exemple de la chaîne fonctionnelle pneumatique, celle-ci comporte deux capteurs de fin de course associés au vérin, et comme ces deux capteurs dépendent seulement de cette chaîne fonctionnelle, la modélisation de leur comportement sera associée à ce seul module. Nous introduisons pour cela, des variables qui représentent ces capteurs et dont les variations des valeurs logiques traduisent le comportement souhaité.

Ainsi, les variables «in» et «out» qui représentent les deux capteurs seront assignées à la valeur logique «1» quand les places «l_in» et «l_out», respectivement, seront actives (figure 21).

3.1.2.3 Temporisations logiques et événements associés

Une difficulté fréquemment rencontrée dans l'élaboration du modèle d'une chaîne fonctionnelle est présente dans la transition entre les places «l_going_out» et «l_out». En effet, si l'ordre «GO_OUT» est maintenu pendant un temps suffisant pour que la chaîne fonctionnelle ait le comportement décrit par place «l_going_out» (sortie de la tige), la tige du vérin finira par arriver en fin de course. Dans ces conditions, le modèle doit changer d'état et activer la place «l_out».

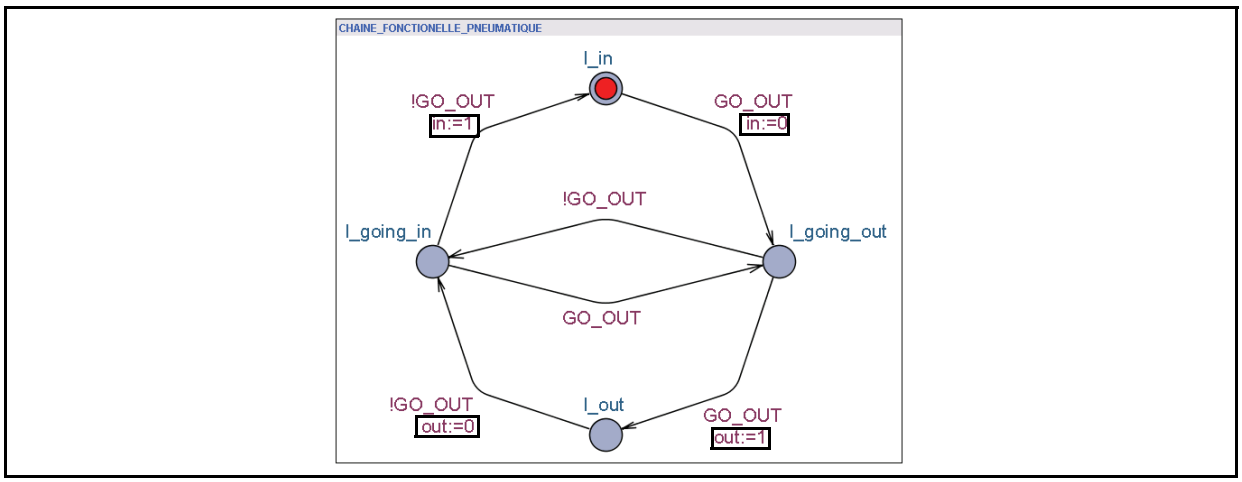


Figure 21: Affectation des variables du modèle correspondant aux capteurs de la chaîne fonctionnelle.

Pour que cela arrive il est nécessaire qu'un événement permette le franchissement de la transition entre les places «l_going_out» et «l_out». En l'absence d'événement observable par le processus (le capteur de fin de course est en effet une sortie du modèle de processus), cette transition doit être franchie après que le temps estimé nécessaire au mouvement se soit écoulé (ce temps dépend directement de la géométrie du vérin et des conditions de travail, comme par exemple la pression, le débit, ...). Comme dans nos modèles nous ne modélisons pas le temps sous une forme physique, mais seulement logique, nous modéliserons ces durées des mouvements au travers d'événements, auxquels nous donnerons le nom de «*Ei*» avec $i \in \mathbb{N}$, et qui représentent que le temps nécessaire à la sortie de la tige du vérin s'est écoulé. Nous avons choisi de ne pas fixer à priori de valeur constante à ces durées opératives et l'apparition de ces événements est donc aléatoire, à partir de l'activité de la place amont à cette transition. Ainsi, notre modèle permet que la durée des mouvements de la tige du vérin soit très courte (mais non nulle), ou très longue. La figure 22 présente deux de ces événements qui modélisent des temporisations logiques (entre les places «l_going_out» et «l_out» et entre les places «l_going_in» et «l_in») :

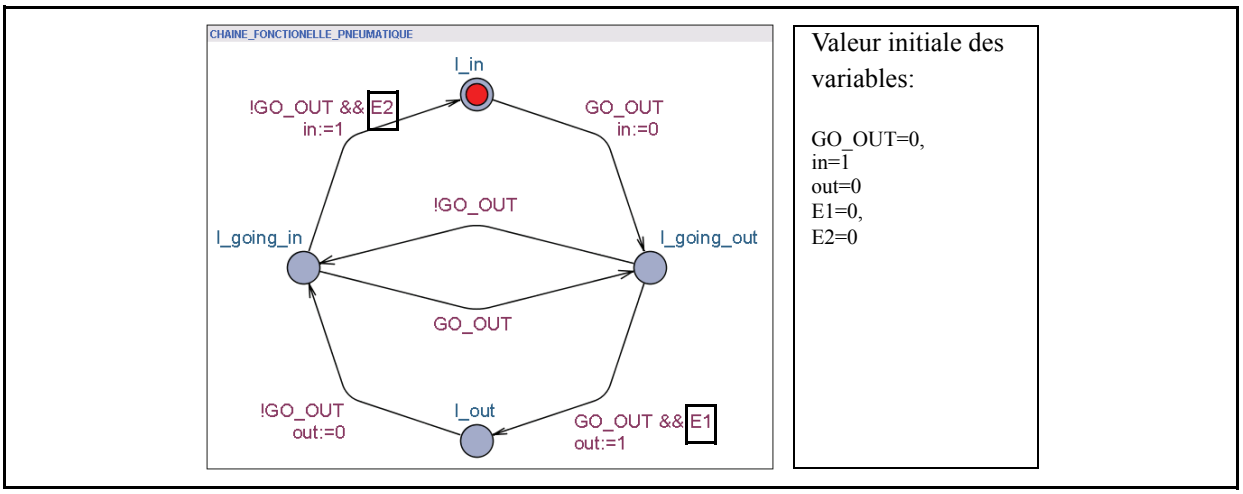


Figure 22: Modèle complet de la chaîne fonctionnelle de la figure 17.

La modélisation des variables Ei sera présentée en détail dans la section "Modélisation du temps logique", page 43.

3.1.3 Modélisation de plusieurs chaînes fonctionnelles utilisant des composants communs

Dans l'exemple de la figure 17, tous les composants du processus appartiennent à une seule chaîne fonctionnelle, mais très souvent certains composants du processus sont «partagés» par plusieurs chaînes fonctionnelles. Considérons par exemple le processus de la figure 23 composé de deux chaînes fonctionnelles (les deux sous-ensembles pneumatiques) et qui comporte un capteur qui dépend des deux chaînes (le capteur «S»). Ce capteur est activé quand les tiges des deux vérins sont complètement sorties.

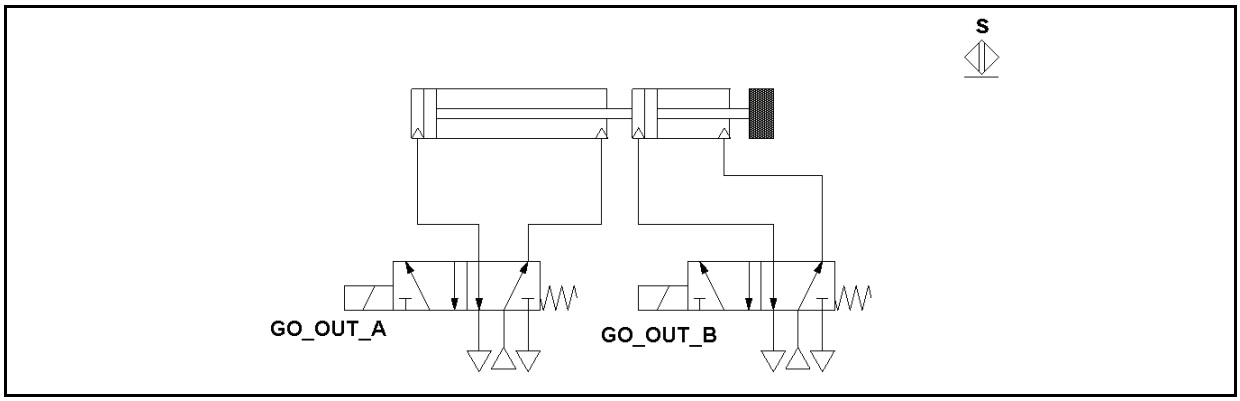


Figure 23: Processus composé de deux chaînes fonctionnelles pneumatiques et d'un capteur dépendant de ces deux chaînes.

Ce que nous proposons, pour la modélisation de processus tels que celui présenté dans la figure 23, est de construire un module pour chacune des deux chaînes fonctionnelles et un module indépendant pour le capteur (figure 24 et figure 25).

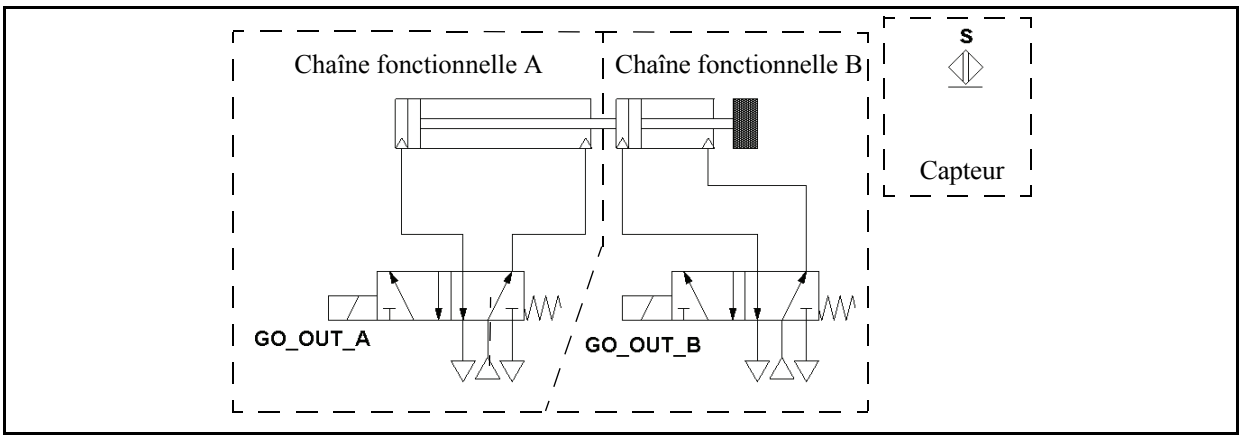


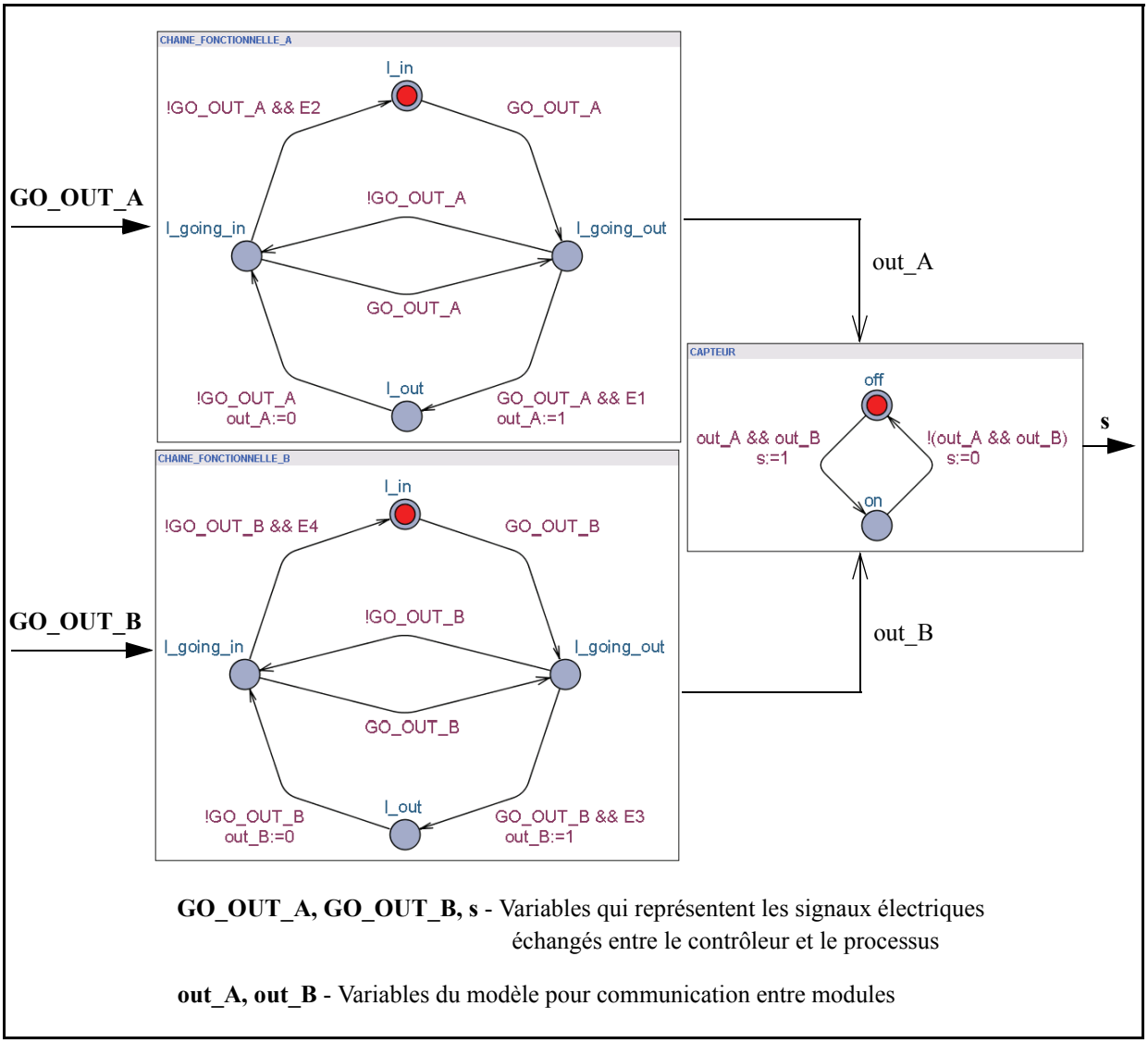
Figure 24: Analyse du processus de la figure 23, en considérant les chaînes fonctionnelles qui le composent.

Comme le comportement du capteur dépend, dans ce cas, des deux chaînes fonctionnelles, cela signifie que le modèle de chaque chaîne fonctionnelle doit interagir avec le modèle du capteur. Chacun de ces trois modèles devant être, conformément à notre objectif de modularité, une instance de modèles de modules génériques se trouvant dans une bibliothèque de composants, incrémentée à chaque construction de modèle pour un nouveau module.

C'est ce que l'on retrouve pour la modélisation des deux chaînes pneumatiques qui sont deux instances du modèle déjà présenté dans la figure 22. Pour le modèle du capteur, on a opté pour un modèle comportant deux places seulement : une place qui représente le capteur non activé «off» et autre place qui représente le capteur activé «on». L'évolution entre ces deux places dépend de l'état d'évolution des modèles des chaînes fonctionnelles dans les places qui modélisent les tiges des deux vérins en position sortie.

Pour ce faire, des variables de communication sont ajoutées au modèle : «out_A» et «out_B». Leur valeur logique est égale à «1» lorsque la place à laquelle elle sont associée est active. Dans la suite de ce mémoire, nous les appellerons «variables du processus» car leur valeur logique «1» est toujours associée à l'activité d'un état du modèle.

Au modèle présenté dans la figure 25 il manque encore les variables E1, E2, E3 et E4, toutes du type Ei défini précédemment, qui permettent de traduire les temps opératifs. Nous allons maintenant en détailler la modélisation.



GO_OUT_A, GO_OUT_B, s - Variables qui représentent les signaux électriques échangés entre le contrôleur et le processus

out_A, out_B - Variables du modèle pour communication entre modules

Figure 25: Modélisation du processus présenté figure 24.

3.1.4 Modélisation du temps logique

Le modèle basique que nous proposons pour la prise en compte du temps logique (variables Ei, dans nos modèles) est basé sur la définition normalisée des blocs «Time On Delay» (TON) [IEC 1993] et sur la modélisation qu'en a proposé O. Rossi dans [ROSSI 2004]. Considérons le modèle présenté dans la figure 22 et détaillons le comportement souhaité pour la variable E1. Chaque fois que la place «l_going_out» n'est pas active, cette variable doit avoir la valeur logique «0». Dès que la place «l_going_out» est active, le temps commence à s'écouler, mais sans changer pour l'instant, la valeur logique de E1 qui reste à la valeur logique «0». Par la suite, si la place «l_going_out» continue à être active, la variable pourra évoluer vers la valeur logique «1». Si, à tout instant, la place «l_going_out» cesse d'être active, la variable doit passer immédiatement (ou rester) à la valeur logique «0».

La création d'un module pour cette variable E1, dont l'évolution dépend d'un module déjà existant, va nécessiter un échange de variables. Il faut en effet mettre en place une nouvelle variable pour la communication entre le modèle de la chaîne fonctionnelle et le modèle de la variable E1. Elle est désignée «V_l_going_out» et sa valeur logique est associée à la place «l_going_out»; Il s'agit également d'une variable d'état. A chaque fois que le franchissement d'une transition active la place «l_going_out», «V_l_going_out» prend la valeur logique «1» et à chaque fois qu'une transition désactive cette même place (figure 26), la variable prend la valeur logique «0». Cette variable permet une communication entre les modules du modèle global et il est possible d'élaborer le module «TEMPO_E1» qui modélisera le comportement de la variable E1.

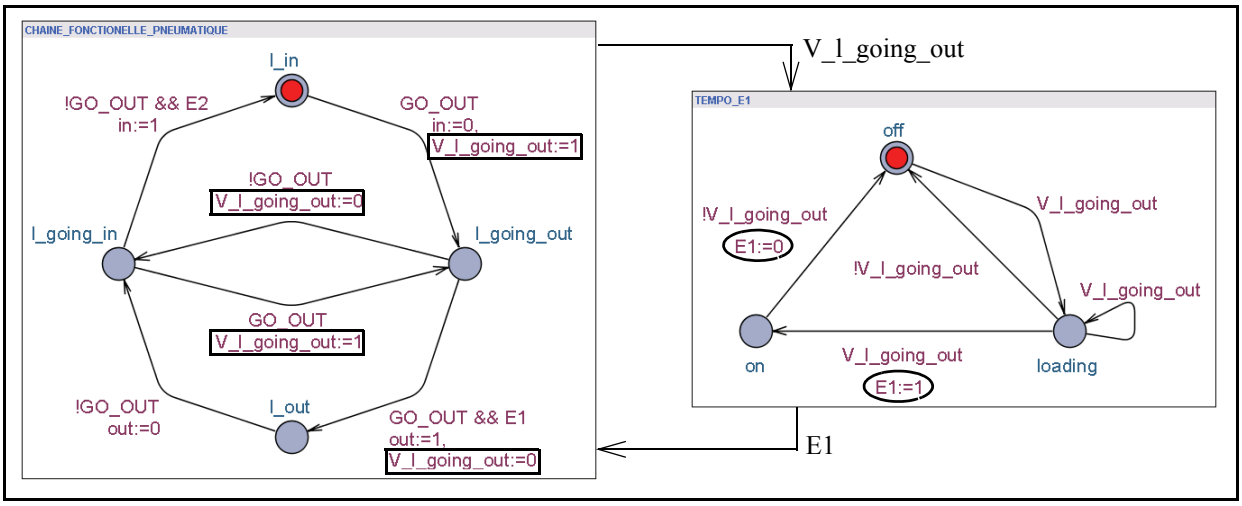


Figure 26: Temporisation logique E1 dans le modèle de la chaîne fonctionnelle présentée figure 17.

Le module correspondant à la temporisation logique est constitué de trois places («*off*», «*loading*» et «*on*»). Chaque fois que la place «l_going_out» est active, «V_l_going_out» vaut«1», le module «TEMPO_E1» a alors la possibilité d'évoluer vers la place «loading». Dans cette place la variable E1 a encore la valeur logique «0», donc le module correspondant à la chaîne fonctionnelle ne pourra pas évoluer. Si la place «l_going_out» reste active («V_l_going_out» reste à «1») le modèle «TEMPO_E1» pourra évoluer vers la place «on» ou pourra rester dans la même place; s'il évolue vers la place «on», la variable E1 aura la valeur logique «1», ce qui permettra l'évolution du modèle de la chaîne fonctionnelle. Dés que la place «l_going_out» cesse d'être active («V_l_going_out» est égal à «0»), quelque soit la place active du module «TEMPO_E1», le module «TEMPO_E1» évoluera vers la place «off», et en conséquence la valeur logique de E1 passera de «1» à «0». Toutes les variables «Ei» auront un modèle de ce type dont une vue générique est donnée figure 27. Dans ce modèle, «V_LOCATION» est la variable associée à une place du modèle du processus à laquelle il faut associer une temporisation logique de type Ei figure 27.

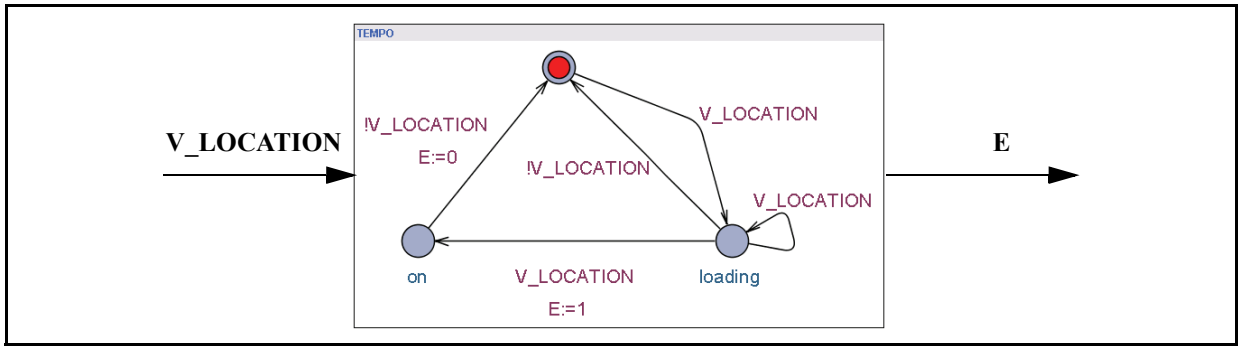


Figure 27: Modèle générique de comportement d'une variable Ei (temps logique).

3.2 Association de modules

Dans la section précédente nous avons montré l'intérêt d'une modélisation modulaire pour la réutilisabilité dans un contexte d'ingénierie de systèmes complexes. Un modèle de processus va donc avoir une structure globale telle que celle représentée dans la figure 28.

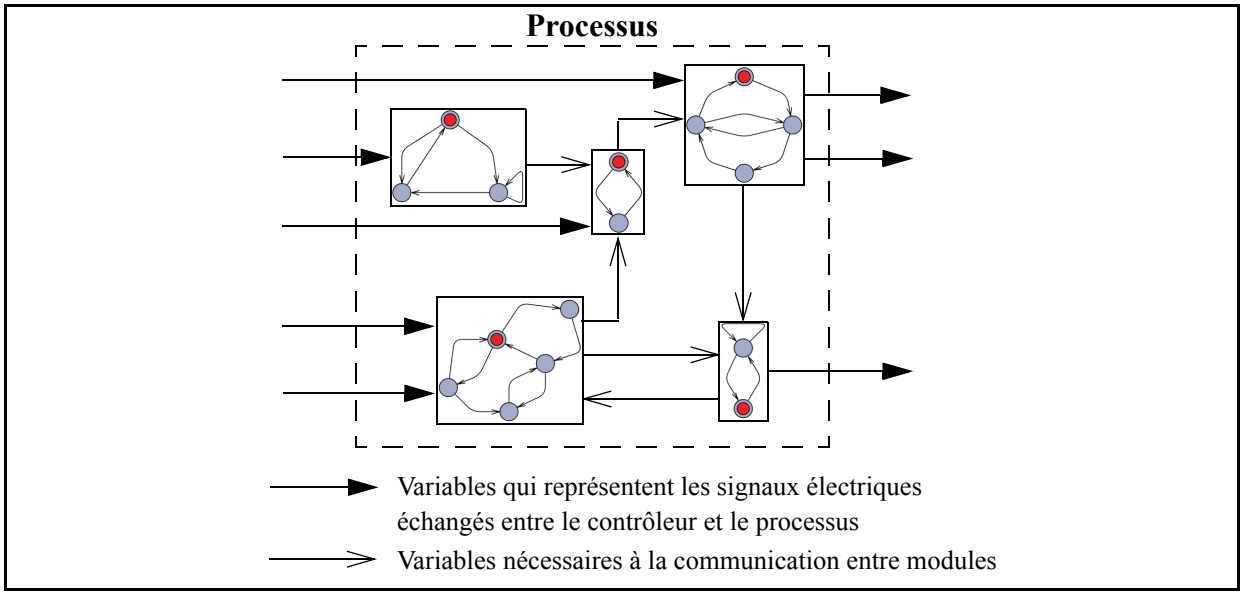


Figure 28: Structure générale d'un modèle du processus

Lorsque les entrées évoluent, les conséquences de ces évolutions se répercutent sur les sorties du processus. Ce phénomène de propagation a pour effet une succession d'évolutions de l'activité des places des modules avant d'arriver à une «situation stable». Seules les situations stables du modèle du processus ont une sémantique vis-a-vis du processus lui-même. Les situations intermédiaire sont dues à la granularité du modèle qui permet la coordination entre modules.

Par exemple, dans le processus présenté figure 23, dont nous avons présenté le modèle figure 25, nous avons la possibilité d'avoir simultanément les places «l_out» (modèle de la chaîne fonctionnelle A), «l_out» (modèle de la chaîne fonctionnelle B) et «off» (modèle du capteur du processus) actives. Cette activité simultanée de trois places constitue une situation qui n'est pas stable parce qu'elle ne correspond pas à un comportement réel du processus. En effet, seule la situation stable «l_out» (modèle de la chaîne fonctionnelle A), «l_out» (modèle de la chaîne fonctionnelle B) et «on» (modèle

du capteur du processus) a un sens. Il est donc important de nous assurer que nous atteindrons toujours une situation stable sur occurrence d'une évolution d'entrées.

La conception d'un modèle modulaire du processus passe par la résolution de ce problème de propagation des événements d'entrée jusqu'à la stabilisation du modèle. Pour illustrer la manière dont nous avons résolu ce problème, nous nous appuierons sur l'exemple du processus modélisé figure 29. Il s'agit d'un processus fictif retenu pour sa concision.

Le processus choisi a le comportement suivant. L'état initial est caractérisé par les places (A, C, E). Le module M1 peut évoluer vers la place B et immédiatement après le module M2 évolue vers la place D, puis le module M3 évolue vers la place F, et ainsi de suite... Les variables V1 et V2 sont les entrées du processus et les variables V_D et V_E sont les sorties du processus. Les autres variables sont des variables de communication entre modules qui constituent le processus.

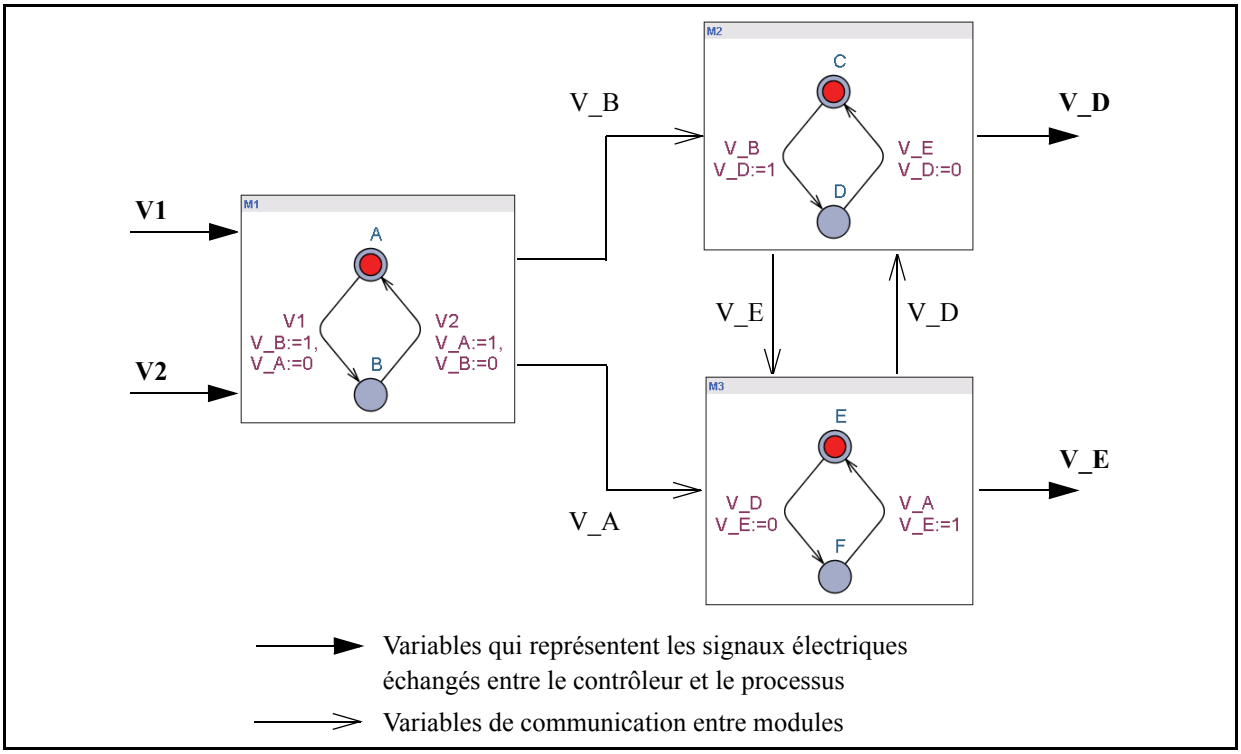


Figure 29: Modules interdépendants qui constituent un processus.

Dans ce processus il y a seulement deux états caractéristiques du comportement du processus : les états *stables* (A, C, E) et (B, D, F) (voir "Stabilité du modèle du processus", page 47). Des états intermédiaires instables, comme par exemple (B, C, E), qui n'ont pas de signification physique sont cependant atteignables et ne représentent aucun comportement pertinent du processus. Pour traduire l'interaction entre les modules d'un même processus et leur stabilisation pour traduire les actions opératives, nous avons défini un séquenceur de processus qui va maintenant être détaillé.

3.2.1 Séquenceur du processus

En effet, comme notre modélisation du processus a comme finalité la preuve de propriétés de comportement par model-checking, un point important à prendre en compte est que la preuve de propriétés ne doit pas être faite dans les états transitoires «produits» par l'utilisation de l'approche modulaire que nous avons retenue, mais seulement dans les états du modèle qui représentent un comportement réel du processus. Puisque le modèle du processus évolue en plusieurs pas (propagation des événements d'entrée) jusqu'à atteindre un état correspondant à une situation réelle du processus, il ne doit pas y

avoir variation de prise en compte des entrées du modèle du processus (sorties du contrôleur) pendant cette évolution jusqu'à ce qu'elle soit terminée.

Nous avons donc défini un module qui va superviser l'évolution du modèle du processus. Nous donnerons à ce module le nom de «*Séquenceur du processus*» (dans les modèles on utilisera la notation «*SEQ_PLANT*»; simplification de la terminologie anglaise «*SEQuencer of the PLANT*»). La figure 30 présente ce module. La place «*PIR*» («*Plant Inputs Reading*») représente la phase de prise en compte des évolutions issues du contrôleur et modélise le début de l'évolution du processus. La place «*PTR*» («*Plant TReatment*») représente la phase de propagation des événements dans les modules qui modélisent le processus. Finalement, la place «*POU*» («*Plant Outputs Updating*») modélise la fin de l'évolution du processus; c'est à dire un état pour lequel les modules du processus représentent une situation réelle du processus.

Une boucle de transition sur PTR a été introduite pour permettre les évolutions de plusieurs modules du processus dans le même état du séquenceur.

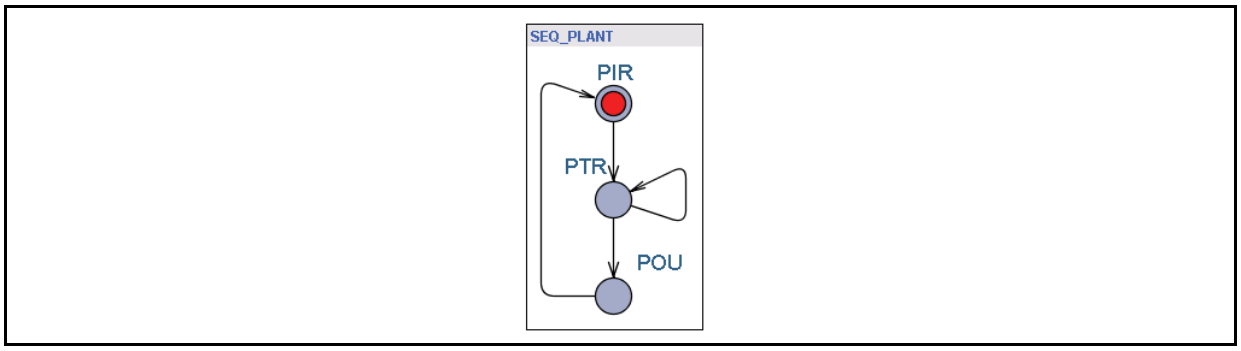


Figure 30: Séquenceur de processus.

3.2.2 Communication entre séquenceur et modules du processus

Prenons un exemple pour illustrer comment faire évoluer un modèle du processus avec ce séquenceur (figure 31). Pour que le processus (M1, M2 et M3) évolue, le module du séquenceur du processus doit se trouver dans la place PTR. Soit «V_PTR» la variable logique qui est vraie lorsque la place PTR est active et fausse lorsque la place PTR est inactive, pour que le processus évolue seulement quand le séquenceur du processus se trouve dans la place PTR, toutes les gardes associés à toutes les transitions des modules du modèle du processus doivent être modifiés en ajoutant «&& V_PTR» à chaque garde (figure 31).

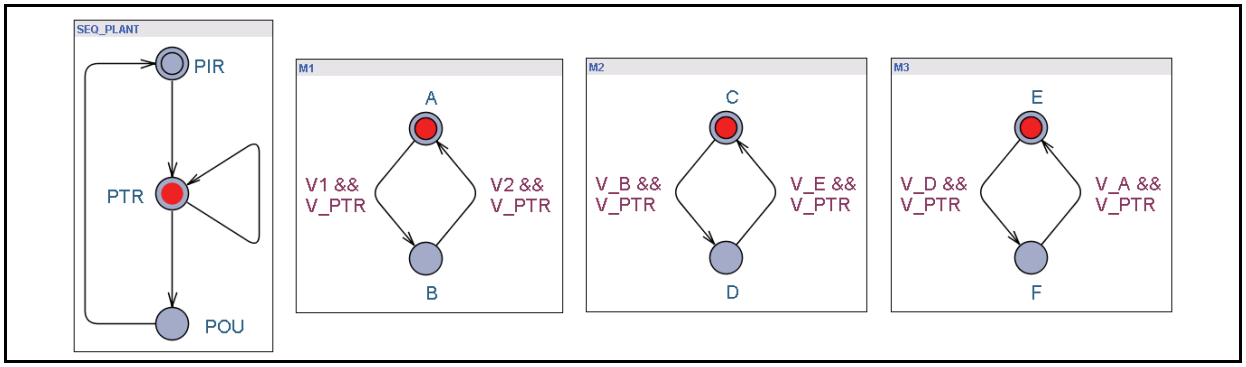


Figure 31: Évolution du processus uniquement lorsque la place PTR est active.

3.2.3 Stabilité du modèle du processus

Comme notre formalisme d'automates est non déterministe, toute transition dont la garde est vraie peut être franchie d'une façon complètement aléatoire. Aussi, nous devons garantir que la transition entre les places PTR et POU ne sera franchie que lorsque le modèle du processus aura atteint un état qui caractérise un état réel du processus (ce que nous avons nommé un état «*stable*» du processus). Un état *stable* du modèle du processus est défini comme un état depuis lequel le modèle ne peut évoluer sans qu'il y ait une variation de ses entrées (sorties du contrôleur). La procédure que nous avons adoptée pour faire évoluer le modèle du processus d'un état *stable* vers un autre état *stable* est la suivante :

1 - chaque module évolue une fois à chaque pas de recherche de stabilité,
2 - si, après un pas, un module a changé de situation, un autre pas de recherche de stabilité est nécessaire,
3 - si, après un pas, aucun module n'a changé de situation, la stabilité est atteinte et le module SEQ_PLANT peut évoluer vers la place POU.

Pour que l'évolution du comportement des automates soit conforme à cette procédure, il a fallu adapter les modules du modèle du processus. Ainsi, lorsque la place PTR est active, tous les modules du modèle du processus doivent avoir une transition franchissable, que cela soit d'une place vers une autre (changement de situation) ou d'une place sur elle-même (stabilité locale du module). Cela conduit à ajouter à chaque place une boucle de transition dont la garde associée est le complément logique des gardes des transitions qui permettent de sortir de cette place (figure 32).

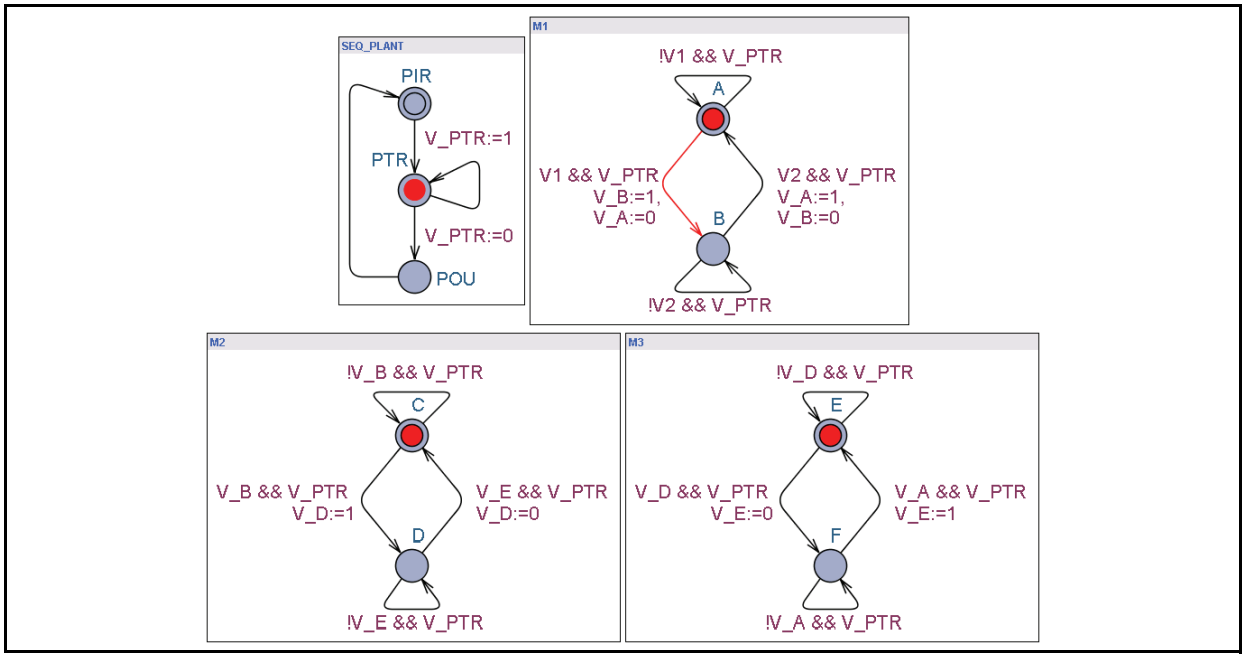


Figure 32: Ajout d'une transition de chaque place sur elle-même pour permettre que chaque module puisse évoluer à chaque pas de recherche de stabilité.

Ensuite, pour être certain que le modèle, depuis l'état stable de départ, change toujours d'état jusqu'à obtenir un état stable nous avons ajouté deux variables (figure 33):

- la variable «*CPL*» («*Change of a Plant Location*») qui nous indique si dans une évolution du modèle du processus il y a eu, ou non, changement de situation. À cette variable est assignée la valeur logique «1» à chaque fois qu'une transition entre deux places différentes est franchie dans un module pendant un pas de recherche de stabilité ;

- la variable «*FMT_i*» («*Firing of a Module Transition*»), où *i* représente le nom du module, à laquelle est assignée la valeur logique «1» à chaque fois qu'une transition de ce module est franchie. L'expression «&& !FMT_i» doit être ajoutée à chaque garde du module *i*, pour que celui-ci ne puisse évoluer qu'une seule fois, pour chaque pas de recherche de stabilité. De la sorte, le module *i* ne pourra plus évoluer une autre fois sans que soit assignée à cette variable la valeur logique «0» (dans un nouveau pas de recherche de stabilité du modèle du processus).

La garde associée à la transition entre les places PTR et POU du séquenceur du processus prend en compte le fait que la valeur de toutes les variables FMT_i est à «1» et que la valeur de la variable CPL est à «0». L'évolution du séquenceur du processus doit garantir que jusqu'à une nouvelle évolution du modèle du processus, les variables FMT_i restent assignées à la valeur logique «0». Il faut donc dans la transition entre les places PIR et PTR, du séquenceur du processus, que soit assignée à toutes les variables FMT_i la valeur logique «0». Dans cette transition la valeur logique «0» est également assignée à la variable CPL, due à la première évolution du séquenceur du processus.

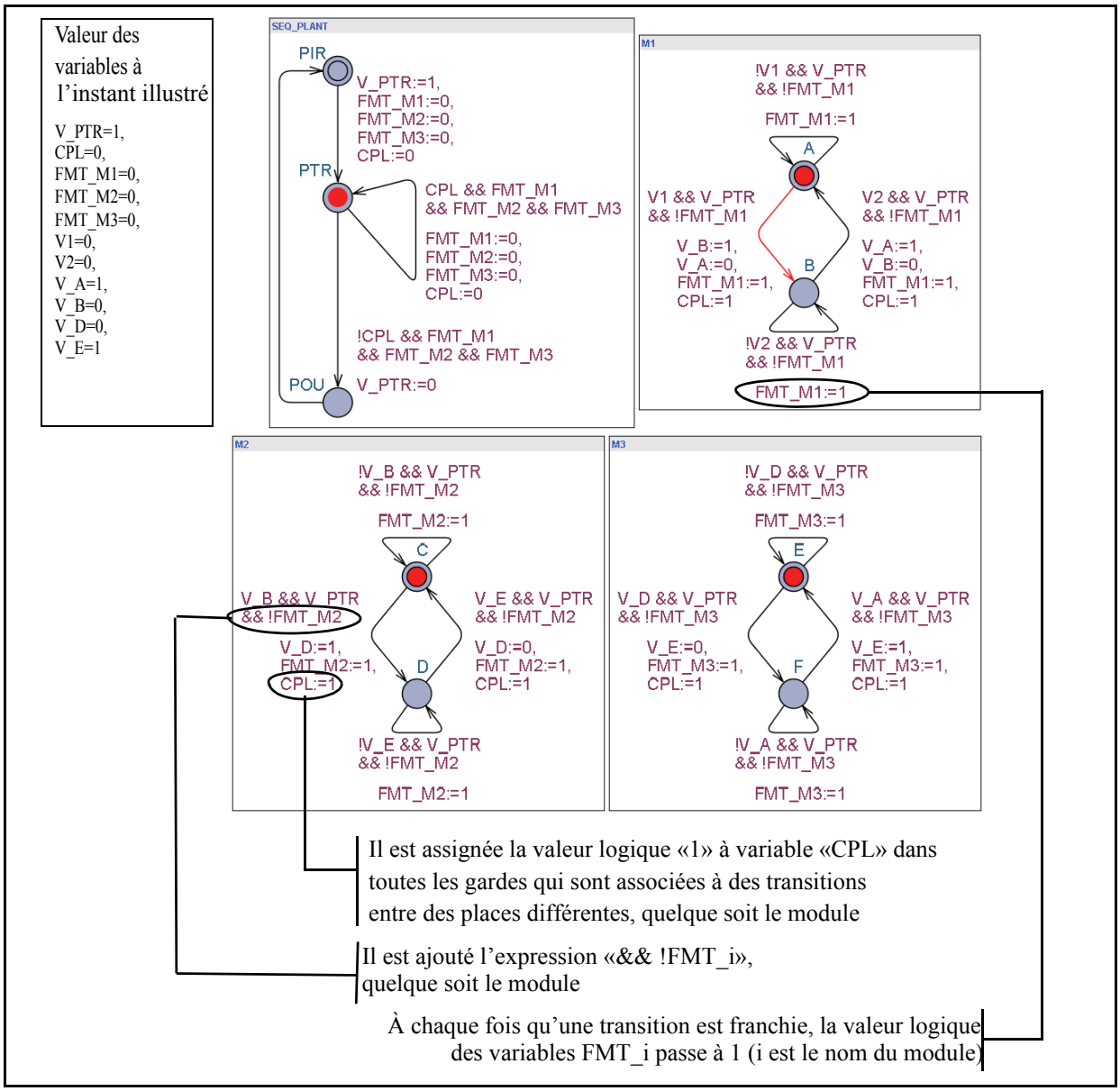


Figure 33: Variables ajoutées au modèle du processus.

On peut résumer la modélisation du processus (avec recherche de stabilité) à l'application des règles suivantes :

- Pour chaque modèle d'un module du processus on doit :
  - Créer une transition bouclée pour chaque place dont la garde est le complément logique des gardes des transitions qui en partent.
  - Ajouter à toutes les gardes l'expression booléenne «&& V_PTR», où V_PTR est la variable associée à l'activité de la place PTR du séquenceur du processus.
  - Créer une variable FMT_i «Firing a Module Transition» (où i est le nom du module). À chaque fois qu'il y a une évolution du module, la variable FMT_i doit être mise à «1». Dans toutes les gardes de toutes les transitions, on doit ajouter l'expression booléenne «&& !FMT_i».
  - Créer une variable CPL «Changing of a Plant Location». À chaque fois qu'un module change de place, la variable CPL doit être mise à «1».
- Dans le séquenceur du processus, on doit :
  - Associer la valeur logique de la variable V_PTR à l'activité de la place PTR.
  - Ajouter à la garde de la transition entre PTR et PTR l'expression booléenne résultante du «et» logique entre toutes les variables FMT_i.
  - Ajouter à la garde de la transition entre PTR et PTR l'expression booléenne «&& CPL»
  - À chaque fois que la transition entre PTR et PTR est franchie, les variables FMT_Ei et la variable CPL sont mises à «0».
  - Ajouter à la garde de la transition entre PTR et POU l'expression booléenne résultante du «et» logique entre toutes les variables FMT_i.
  - Ajouter à la garde de la transition entre PTR et POU l'expression booléenne «&& !CPL».

Ces règles peuvent bien entendu être appliquées d'une façon systématique, ce qui nous permet d'envisager une automatisation complète de la modélisation du processus à partir des modèles d'une bibliothèque de modules. Des algorithmes ont été proposés dans ce sens dans [MACHADO *et al.* 2006]

3.2.4 Synchronisation des modèles de temporisation avec le séquenceur du processus

Considérons le modèle présenté dans la figure 26 et étudions la synchronisation entre le modèle de cette variable (module «TEMPO» de la chaîne fonctionnelle présentée dans la figure 17) avec le séquenceur du processus (figure 34).

Le problème de stabilité des modules opératifs que nous venons de traiter ne concerne pas les modules de temporisation. Si nous appliquons à ces temporisations les mêmes règles que celles que nous venons de définir pour les modules opératifs, leur comportement ne sera pas celui attendu. Il est donc nécessaire de dissocier les évolutions des modèles des variables Ei de l'activité de la place PTR du séquenceur du processus.

Nous proposons de faire évoluer les modèles de comportement des variables Ei durant l'activité de la place PIR du séquenceur du processus. Pour cela nous ajoutons l'expression booléenne «&& V_PIR» (variable associée à l'activité de la place PIR) à toutes les gardes de toutes les transitions de tous les modules Ei.

Durant l'activité de cette place on forcera l'évolution de tous les modèles des variables Ei seulement une fois et après l'évolution de chaque modèle de chaque variable Ei (obligatoirement une évolution de chaque modèle d'un module Ei) le séquenceur du processus évolue vers la place PTR. Tous

les modèles correspondant aux modules du processus réel peuvent alors évoluer conformément aux règles précédemment stipulées dans le but d'atteindre un état stable du modèle du processus.

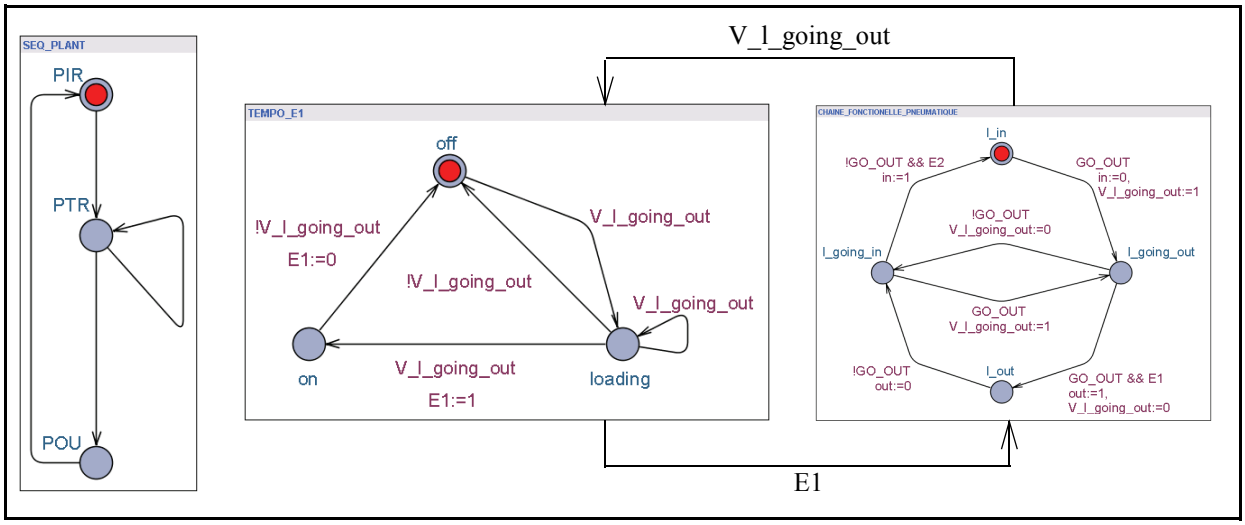


Figure 34: Modèle de la variable E1 (appartenant au modèle de la chaîne fonctionnelle présentée dans la figure 17) et séquenceur du processus.

Pour que l'évolution de toutes les temporisations logiques puissent se faire, il faut ajouter une transition sur la place «off» du modèle initial de la temporisation pour permettre à ce modèle de toujours évoluer vers la place PIR. La garde associée à cette transition est donc le complément logique de la garde associée à la transition qui permet la sortie de cette place (figure 35).

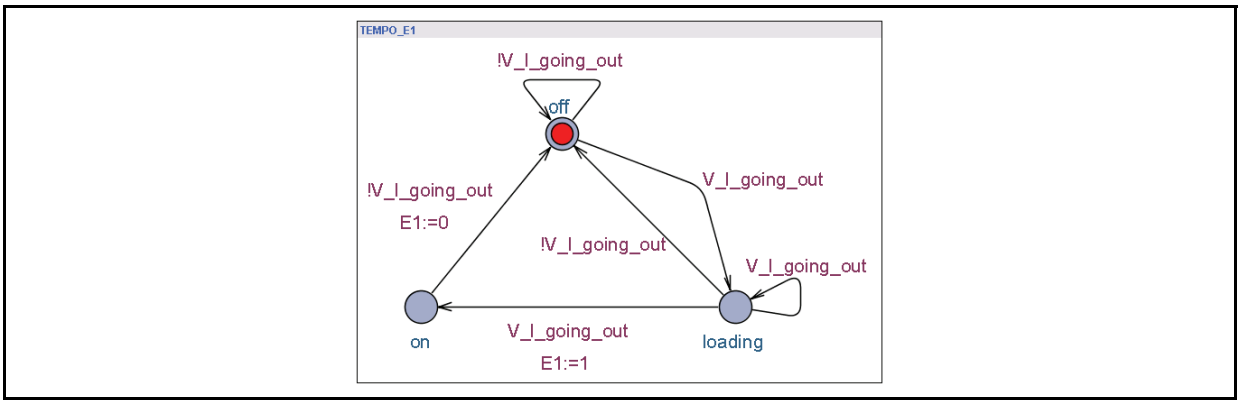


Figure 35: Ajout d'une transition de la place «off» sur elle même.

Pour que le modèle d'une variable Ei puisse évoluer seulement une fois, durant l'activité de la place PIR du séquenceur du processus, nous associerons une variable (FMT_Ei «Firing a Module Transition» du module Ei) à chaque module Ei et cette variable est assignée à la valeur logique «1» à chaque fois qu'une transition de ce module est franchie. Dans le même temps, à chaque garde de chaque transition de chaque module Ei est ajoutée l'expression booléenne «&& !FMT_Ei». Cette expression booléenne fait que chaque module peut évoluer seulement une fois jusqu'à ce que la variable FMT_Ei soit mise à nouveau à «0». Après que les variables FMT_Ei soient mises à «0» tous les modules Ei peuvent évoluer une nouvelle fois. Cette mise à «0» des variables FMT_Ei sera faite dans la transition reliant PIR à PTR. Cela permettra que dans le prochain cycle, les modules Ei puissent évoluer une nouvelle fois lorsque le séquenceur du processus reviendra dans sa place PIR.

La figure 36 illustre la synchronisation entre le modèle de la variable E1 et le séquenceur du processus. L'obtention du modèle de tous les modules Ei est faite de la même manière et peut être résumée par les règles suivantes :

- Dans le modèle générique d'une temporisation logique :
  - La variable V_LOCATION est la variable associée à l'activité de la place qui déclenche la temporisation logique.
  - La variable V_PIR est la variable associée à l'activité de la place PIR du séquenceur du processus.
  - La variable FMT_Ei est une variable associée à chaque module Ei pour permettre une seule évolution du modèle de chaque module, à chaque fois que le séquenceur du processus est dans la place PIR.
  - La variable Ei donne l'information «fin de tempo».
- Dans le séquenceur du processus, on doit :
  - ajouter une variable (V_PIR) à l'activité de la place PIR,
  - ajouter à la garde de la transition entre PIR et PTR une expression booléenne qui est un «et» logique sur toutes les variables FMT_Ei,
  - mettre à «0» toutes les variables FMT_Ei dans les assignations de la transition entre PIR et PTR.

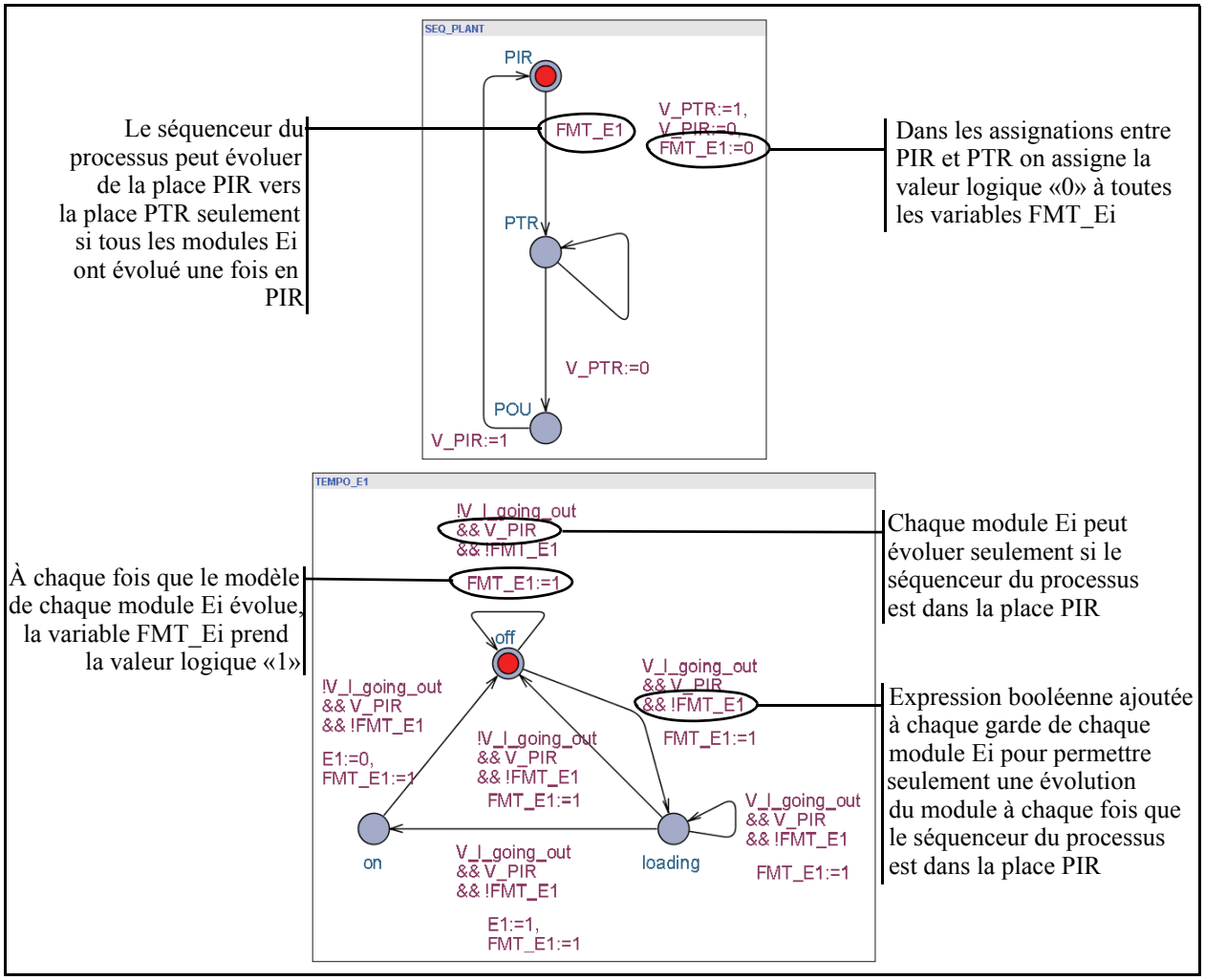


Figure 36: Synchronisation entre un modèle de temporisation et le séquenceur du processus.

On peut donc maintenant proposer un modèle générique pour toutes les temporisation (figure 37).

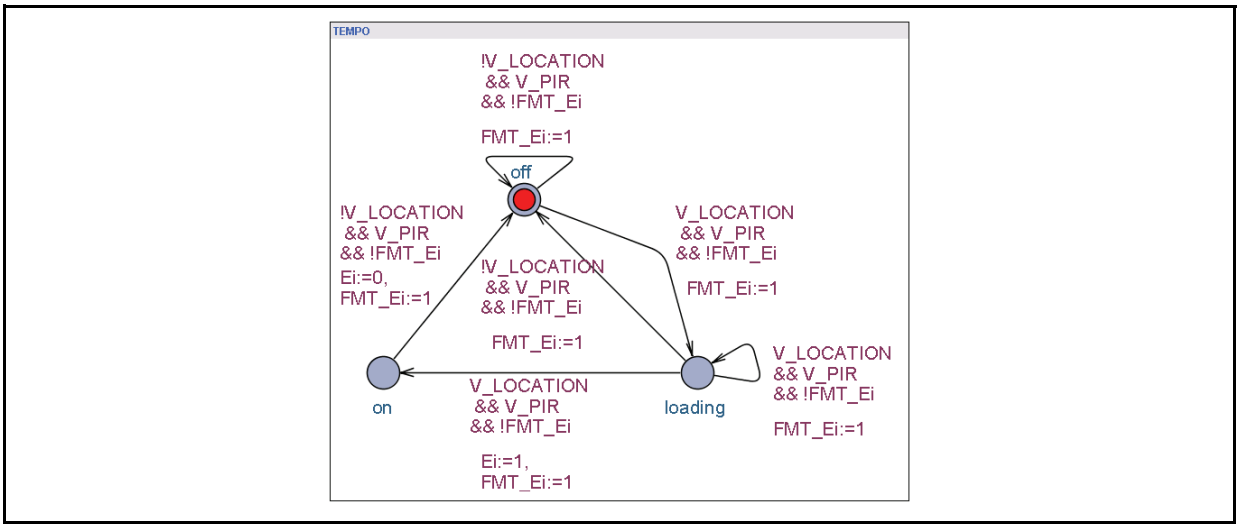


Figure 37: Modèle générique d'une temporisation logique.

3.3 Modèle complet d'un processus composé d'une chaîne fonctionnelle

Considérons le modèle de la figure 26 (en considérant, maintenant, les deux variables qui modélisent les temporisations logiques E1 et E2) et étudions la synchronisation entre le modèle de la chaîne fonctionnelle et le modèle du séquenceur du processus (figure 38). En appliquant les règles présentées en page 46 et page 47 nous obtenons le modèle présenté dans la figure 38.

Après la synchronisation entre le modèle du processus et le séquenceur du processus, il faut définir comment sera faite la synchronisation des modèles des variables Ei avec le modèle global obtenu jusqu'à présent. En appliquant les règles définies page 49 et en appliquant cette démarches aux variables E1 et E2, nous obtenons le modèle global de la chaîne fonctionnelle pneumatique (de la figure 17) qui est présenté dans la figure 39.

On peut constater que pour une simple chaîne fonctionnelle le modèle global obtenu n'est pas trivial, mais rappelons qu'il est élaboré d'une façon systématique conformément aux règles définies dans ce chapitre à partir des modèles génériques contenus dans une bibliothèque.

3.4 Bilan du chapitre 3

Dans ce chapitre nous avons défini des règles pour la modélisation d'un processus que l'on peut appliquer systématiquement. Le choix de modéliser chaque chaîne fonctionnelle par un module est très important dans une optique de vérification formelle. En effet, cela permet d'obtenir des modules compacts, qui traduisent la dynamique dévolution des variables manipulées pour exprimer les propriétés de sûreté et de vivacité, et donc finalement de faciliter la tâche du model-checking. En conséquence, cela nous permettra de vérifier des systèmes de plus grand taille.

La garantie que le processus évolue toujours entre des états stables est fondamentale pour la preuve de propriétés. Cela permet d'identifier clairement quels sont les états du modèle du processus où la preuve de propriétés sera faite. La définition d'un séquenceur de processus (qui est générique, quel

que soit le processus à modéliser) a été une étape importante pour garantir que le modèle du processus évolue systématiquement avec recherche de stabilité.

La modélisation des temporisations logiques et de leur synchronisation avec la séquenceur du processus est également fondamentale pour que nos modèles représentent, d'une façon réaliste, le comportement du processus.

Les procédures systématiques que nous avons proposées permettent d'envisager la modélisation de processus complexes de taille industrielle.

Dans le chapitre suivant, nous allons maintenant présenter un cadre formel pour la vérification d'un système automatisé. Pour cela on va utiliser un exemple support.

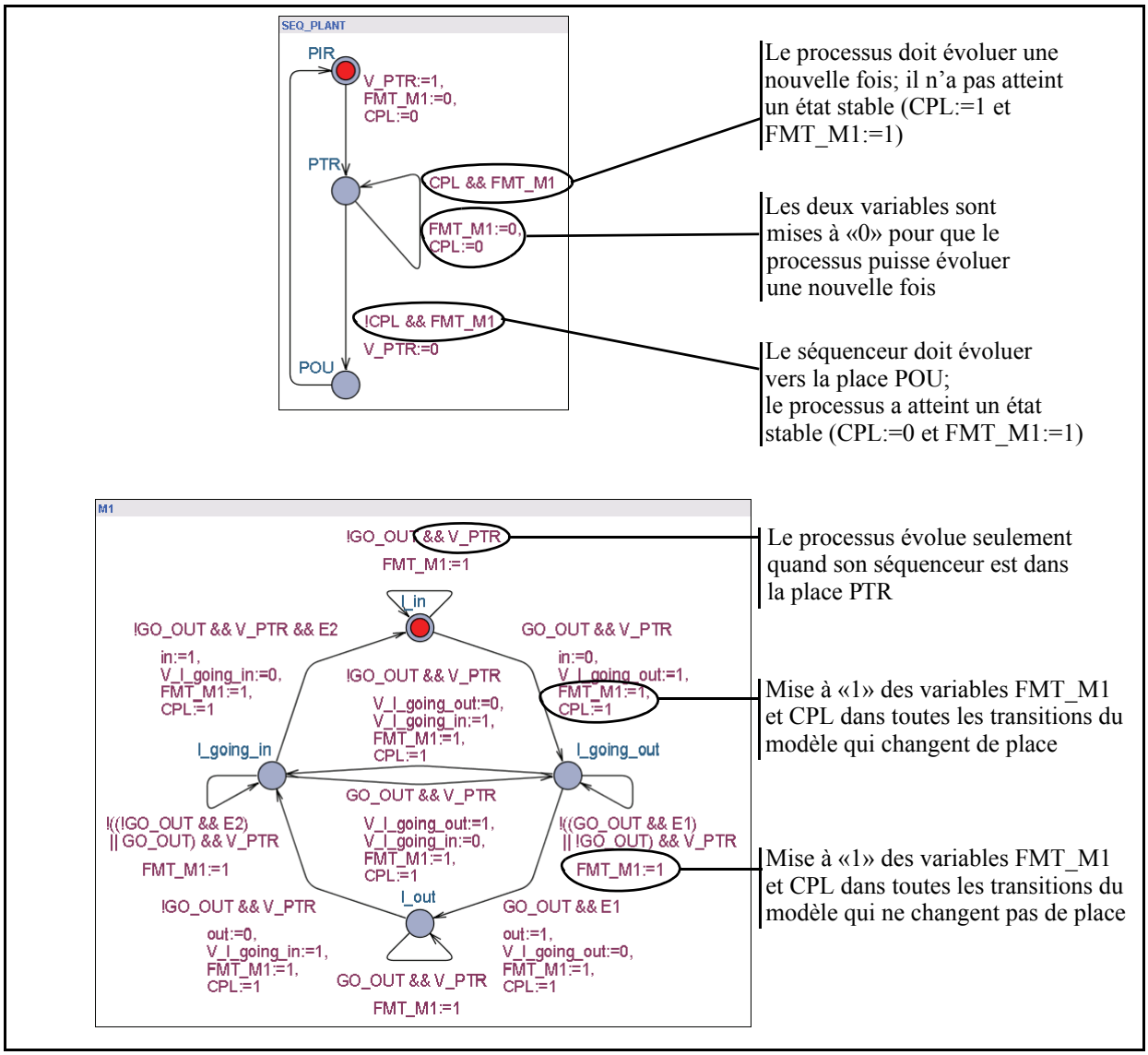


Figure 38: Synchronisation entre le modèle du module correspondant à la chaîne fonctionnelle et le modèle du séquenceur du processus.

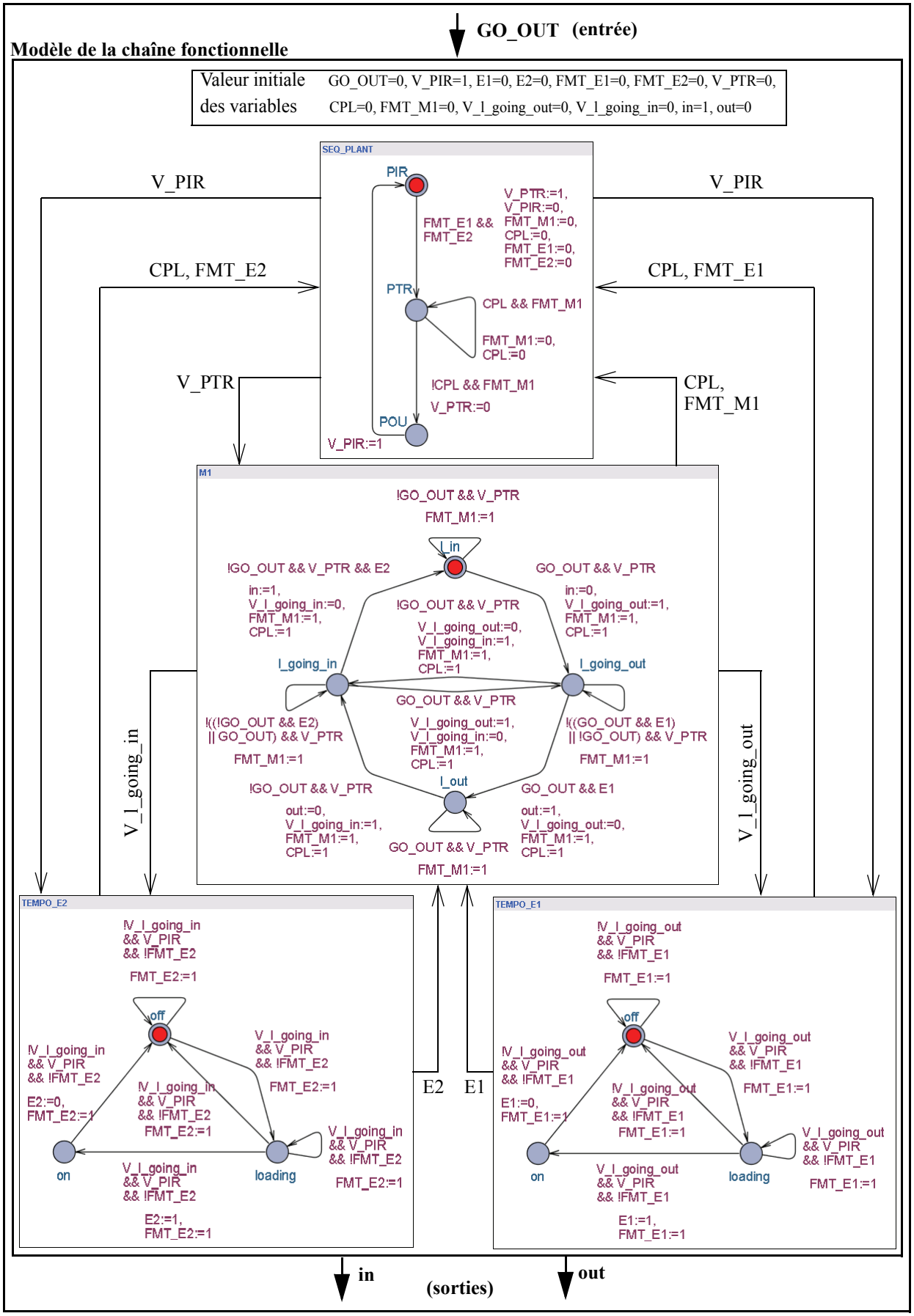


Figure 39: Modèle global final de chaîne fonctionnelle présentée dans la figure 17.

Chapitre 4
Construction du modèle de vérification du système complet

Dans ce chapitre on présente un cadre formel pour la vérification du comportement des systèmes automatisés. Nous présentons d'abord ce cadre d'une manière générique, applicable à tout système automatisé discret, puis nous utilisons un exemple support pour le mettre en oeuvre. Le système automatisé utilisé comme support de notre approche a été construit pour être représentatif de la difficulté de modélisation du système commandé. De même, les propriétés à prouver sont représentatives des différentes classes de propriétés que l'automaticien cherche classiquement à prouver par model-checking.

Dans le chapitre précédent on a présenté en détail comment obtenir le modèle d'un processus et montré la nécessité d'utiliser un séquenceur du processus. Dans ce chapitre nous indiquerons comment obtenir le modèle du contrôleur et de son séquenceur. Nous présenterons également la modélisation des propriétés et de la synchronisation entre les modèles du contrôleur, du processus et des propriétés, que nous appellerons «Séquenceur Général». C'est avec ce séquenceur général qu'on pourra maîtriser l'évolution de tous nos modèles d'une manière adaptée à la réalisation de la vérification formelle.

4.1 Modèle du contrôleur et de son séquenceur

Dans notre approche, nous faisons l'hypothèse que le contrôleur a un comportement cyclique, mais non nécessairement périodique. Nous considérons également que les programmes du contrôleur sont élaborés en syntaxe Sequential Function Chart (SFC - [IEC 1993]).

Le comportement dynamique du programme du contrôleur est la combinaison entre la dynamique propre du programme et le comportement du contrôleur pendant son exécution. Compte tenu de nos hypothèses de travail (contrôleur à moniteur d'exécution cyclique et programme en SFC), nous avons adopté une modélisation efficace et compacte du comportement du programme basé sur des équations algébriques récurrentes, comme développé dans [MARCÉ & LE PARC 1993] et [LAMPÉRIÈRE-COUFFIN & LESAGE 2000].

Avant de rappeler les équations de traduction algébrique d'un SFC, définissons les notations que nous allons utiliser :

- $e_j(t_i)$ est l'état de la variable d'entrée e_j à l'instant t_i. $E(t_i)$ est le vecteur de l'ensemble des entrées à l'instant (t_i).
- l'état (actif ou inactif) d'une étape n à un instant t_i s'écrit Xn (t_i) et vaut «0» ou «1» à chaque instant.
- La situation $\overrightarrow{Q}$ (t_i) d'un SFC à un instant t_i est le vecteur d'état de toutes les étapes : $Q(t_i) = (Xn)(t_i)$. Elle peut également se représenter par la liste des étapes actives.

Avec les notations précédentes, la situation d'un SFC à l'instant t_i, fonction de sa situation à l'instant t_{i-1} s'écrit:

$$Q(t_i) = F(E(t_{i-1}),Q(t_{i-1})) \quad (1)$$

où *F* est la fonction qui résulte des conditions de franchissement des transitions du SFC.

On peut donner une formulation générale de l'évolution de l'état d'une étape quelconque d'un SFC, tenant compte de toutes les règles d'évolution, pour une étape «n» donnée possédant «p» transitions précédentes, notées $p_{j,}$, et «s» transitions suivantes, notées s_k, (figure 40).

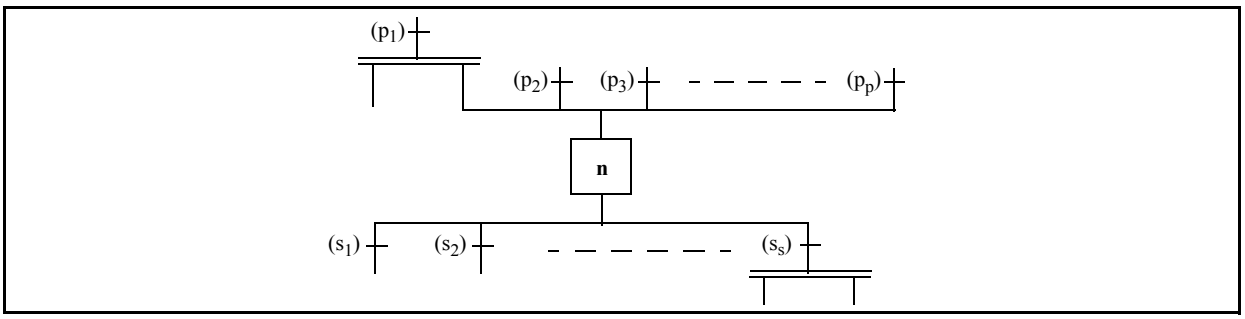


Figure 40: Activité d'une étape du SFC en accord avec la IEC 60848.

$$Xn(t_i) = Cpn(t_i) \vee [Xn(t_{i-1}) \wedge \neg Cns(t_i)] \quad (2)$$

où :

- *Cpn(t_i)* désigne la réunion des conditions de franchissement de toutes les transitions p_j précédant l'étape n, à l'instant t_i

$$Cpn(t_i) = \sum_{j=1}^{p} CF(p_j)(t_i) \quad (3)$$

- *Cns(t_i)* désigne la réunion des conditions de franchissement de toutes les transitions s_k suivant l'étape n, à l'instant t_i

$$Cns(t_i) = \sum_{k=1}^{s} CF(s_k)(t_i) \quad (4)$$

Par ailleurs, la condition de franchissement «*CF(tr)*» d'une transition *tr* de réceptivité «*r*» et validée par m étapes M_j (figure 41) s'exprime algébriquement par le produit logique de l'ensemble des états des étapes M_j et de la réceptivité r :

$$CF(tr)(t_i) = [XM_1(t_i) \wedge XM_2(t_i) \wedge \ldots \wedge XM_m(t_i)] \wedge r \quad (5)$$

$$CF(tr)(t_i) = \left(\prod_{j=1}^{m} XM_j(t_i) \right) \wedge r(t_i) \quad (6)$$

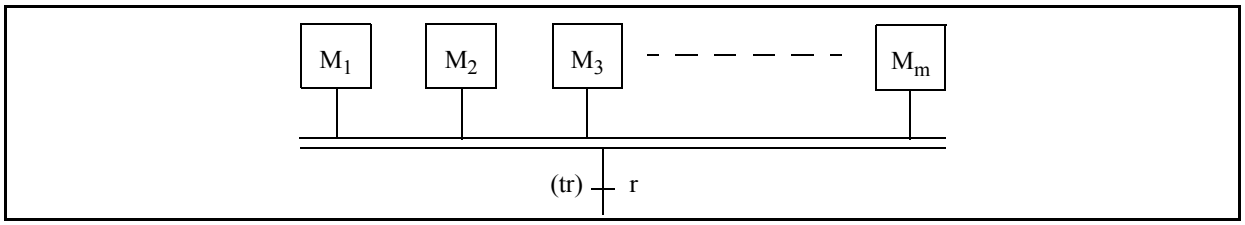


Figure 41: Condition de franchissement après séquences parallèles en accord avec la IEC 60848.

Rappelons enfin que si ES_j est l'ensemble des étapes qui possèdent une action relative à la sortie *S_j(t)* et *si Cd(t)* est la condition d'assignation associée à l'action (quand il n'existe pas de condition d'assignation associée à l'action, cela revient à considérer une condition d'assignation toujours vraie), on a :

$$S_j(t) = \sum_{i \in ES_j} [X_i(t) \wedge Cd_i(t)] \quad (7)$$

Ces équations de traduction algébrique d'un SFC une fois rappelées, il convient de spécifier comment et quand doivent être calculées ces équations dans le modèle du contrôleur. Pour ce faire, considérant un algorithme d'exécution du SFC sans recherche de stabilité (tel que proposé par [BIEREL *et al.* 1997]) et compte tenu du comportement défini précédemment pour nos contrôleurs (présenté dans le chapitre 2), nous avons défini un séquenceur du contrôleur (*«Sequencer of the Controller»*). Grâce à ce séquenceur, toutes les variables (entrées, conditions de franchissement des transitions, variables associées à l'activité d'étape et sorties du contrôleur) peuvent évoluer conformément aux règles d'évolution d'un SFC. Ce séquenceur possède cinq places (figure 42) qui sont:

- La «lecture des entrées»; place «CIR» (*Controller Inputs Reading*). Cette place modélise la prise en compte, par le contrôleur, de ses entrées qui ne seront plus relues avant le prochain cycle du contrôleur. Pour que cette caractéristique comportementale soit garantie par notre

modèle, nous avons décidé de faire une copie de toutes les variables d'entrée (e_i_c : avec e_i la variable d'entrée «*i*» et e_i_c la copie de cette variable d'entrée). Ensuite, dans l'évolution du modèle du contrôleur, seules ces «copies» seront utilisées. Cela garantit que même s'il existe une évolution du modèle du processus qui modifie la valeur logique des variables d'entrée du contrôleur, cette modification ne sera observée par le modèle du contrôleur que dans le cycle suivant.

- Le «franchissement des conditions de transition»; place «CTF» (*Controller Transitions Firing*). La valeur logique donnée par l'équation (6) est assignée aux variables *CF(tr)*, mais en utilisant les copies des entrées (e_i_c) et non pas les entrées elles-mêmes.
- le «traitement du programme»; place «CTR» (*Controller TReatement*). Dans cette place on calcule toutes les variables associées à l'activité des étapes du SFC conformément à l'équation (2).
- l'«évaluation des variables de sortie»; place «COE» (*Controller Outputs Evaluation*). Avant d'émettre ses sorties (équation (7)), le contrôleur calcule la valeur de ses variables en mémoire, qui ont été appelées copies des sorties (S_i_c : avec S_i la variable de sortie «*i*» du modèle du contrôleur et S_i_c la copie de cette variable de sortie).
- l'«écriture des sorties»; place «COW» (*Controller Outputs Writing*) qui modélise l'envoi des ordres du contrôleur au processus.

Dans notre formalisme, tout l'assignement des variables est fait durant le franchissement des transitions (figure 42). Pour l'instant, toutes les gardes du modèle ont la valeur logique «1».

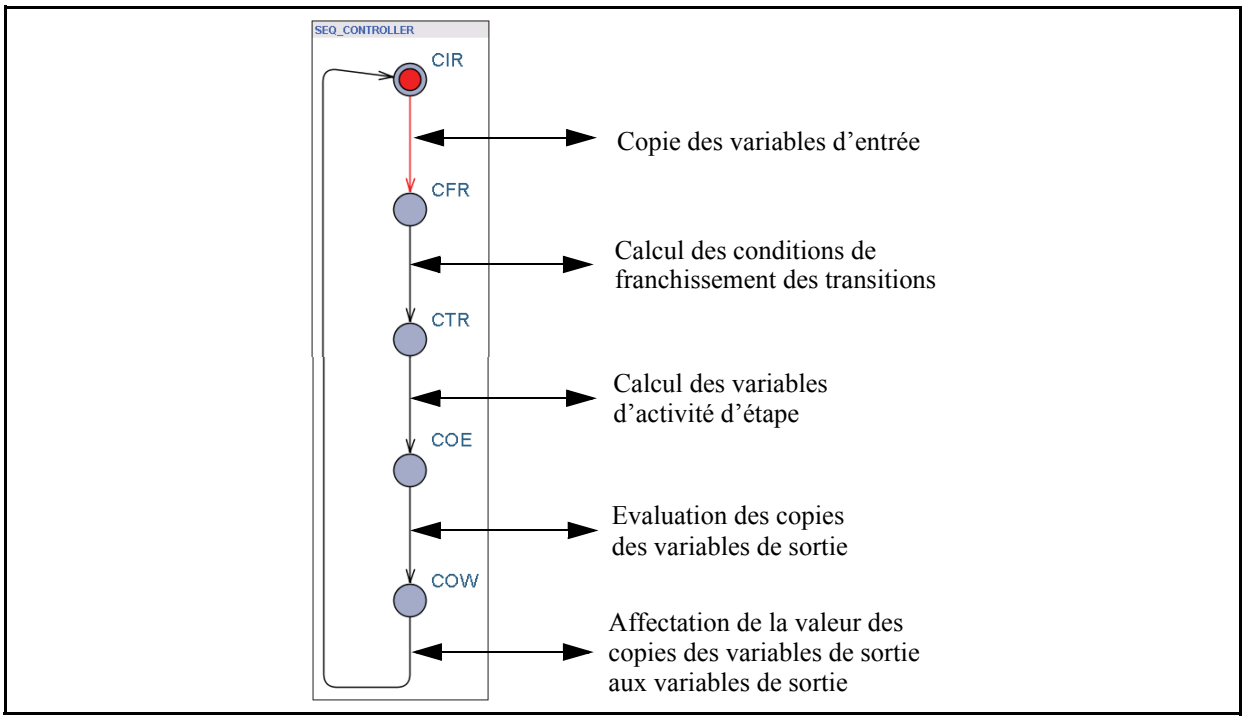


Figure 42: Séquenceur et modèle d'un contrôleur dont le programme est élaboré en SFC.

Si nous ne considérions que l'évolution du contrôleur, le modèle présenté figure 42 serait suffisant, mais on doit maintenant prendre en compte l'interaction de ce modèle du contrôleur avec le modèle du processus et notre objectif de model-checking. Il faut pour cela ajouter au modèle :

- une place - «CTE» (*Controller TEst*) - qui servira au test de propriétés intrinsèques au modèle du contrôleur ou à toutes celles qu'on voudra vérifier après une évolution du contrôleur.

- La possibilité d'avoir une ou plusieurs évolutions du contrôleur par évolution du modèle du processus (figure 43). En effet, le contrôleur doit se comporter de manière infiniment réactive aux évolutions du processus. Il faut donc ajouter une place «CEV» (*Controller EValuation*) au modèle du contrôleur (elle sera placée immédiatement après la place COW), place à partir de laquelle le contrôleur pourra évoluer aléatoirement vers une autre exécution de son cycle sans que le modèle du processus n'ait eu le temps d'évoluer.

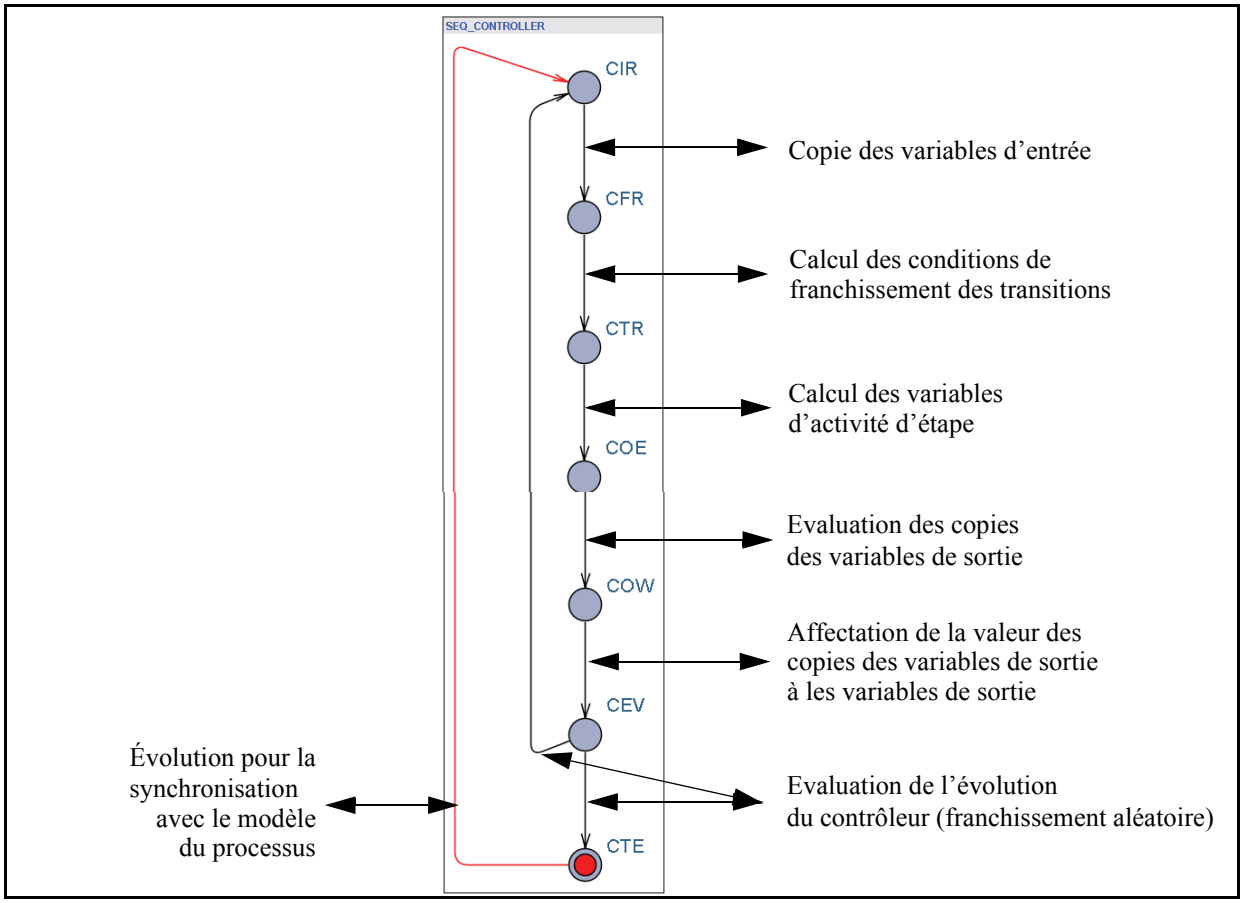


Figure 43: Version finale du modèle du contrôleur en prenant en compte le test de propriétés et sa relation avec le modèle du processus et sa réactivité.

Dans cette version définitive, la place initiale est la place CTE où le test des propriétés sera fait. Le modèle du contrôleur restera dans cette place pendant que les autres modèles évoluent. La synchronisation entre les modèles du contrôleur et du processus sera faite grâce à la garde associée à la transition CTE → CIR. À titre d'exemple, considérons maintenant un contrôleur qui exécute un programme décrit par le SFC de la figure 44.

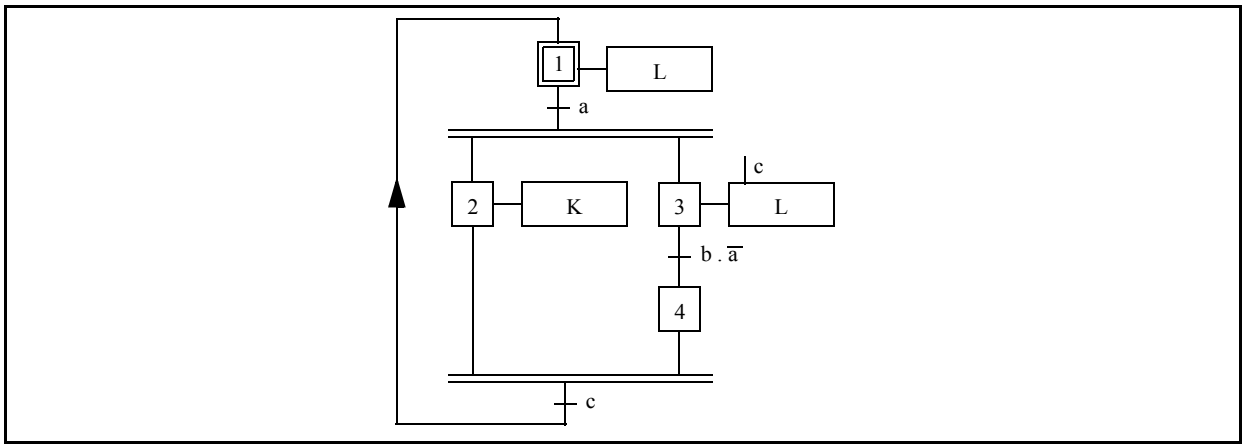


Figure 44: Exemple de spécification d'un contrôleur à modéliser

La traduction algébrique nous donne:

- Pour le calcul des conditions de franchissement (selon l'équation (6)) :

$$CF(1)(t) = X1(t) \wedge a$$
$$CF(2)(t) = X3(t) \wedge (b \wedge \neg a)$$
$$CF(3)(t) = X2(t) \wedge X4(t) \wedge c$$

- Pour le calcul des états d'étapes (selon l'équation (2)) :

$$X1(t) = CF(3)(t) \vee X1(t-1) \wedge \neg CF(1)(t)$$
$$X2(t) = CF(1)(t) \vee X2(t-1) \wedge \neg CF(3)(t)$$
$$X3(t) = CF(1)(t) \vee X3(t-1) \wedge \neg CF(2)(t)$$
$$X4(t) = CF(2)(t) \vee X4(t-1) \wedge \neg CF(3)(t)$$

- Pour le calcul des valeurs assignées aux sorties (selon l'équation (7)) :

$$K(t) = X2(t)$$
$$L(t) = X1(t) \vee (X3(t) \wedge c(t))$$

En intégrant ces expressions algébriques dans le séquenceur du contrôleur, nous obtenons l'automate de la figure 45 qui modélise complètement le comportement de l'exécution du contrôleur.

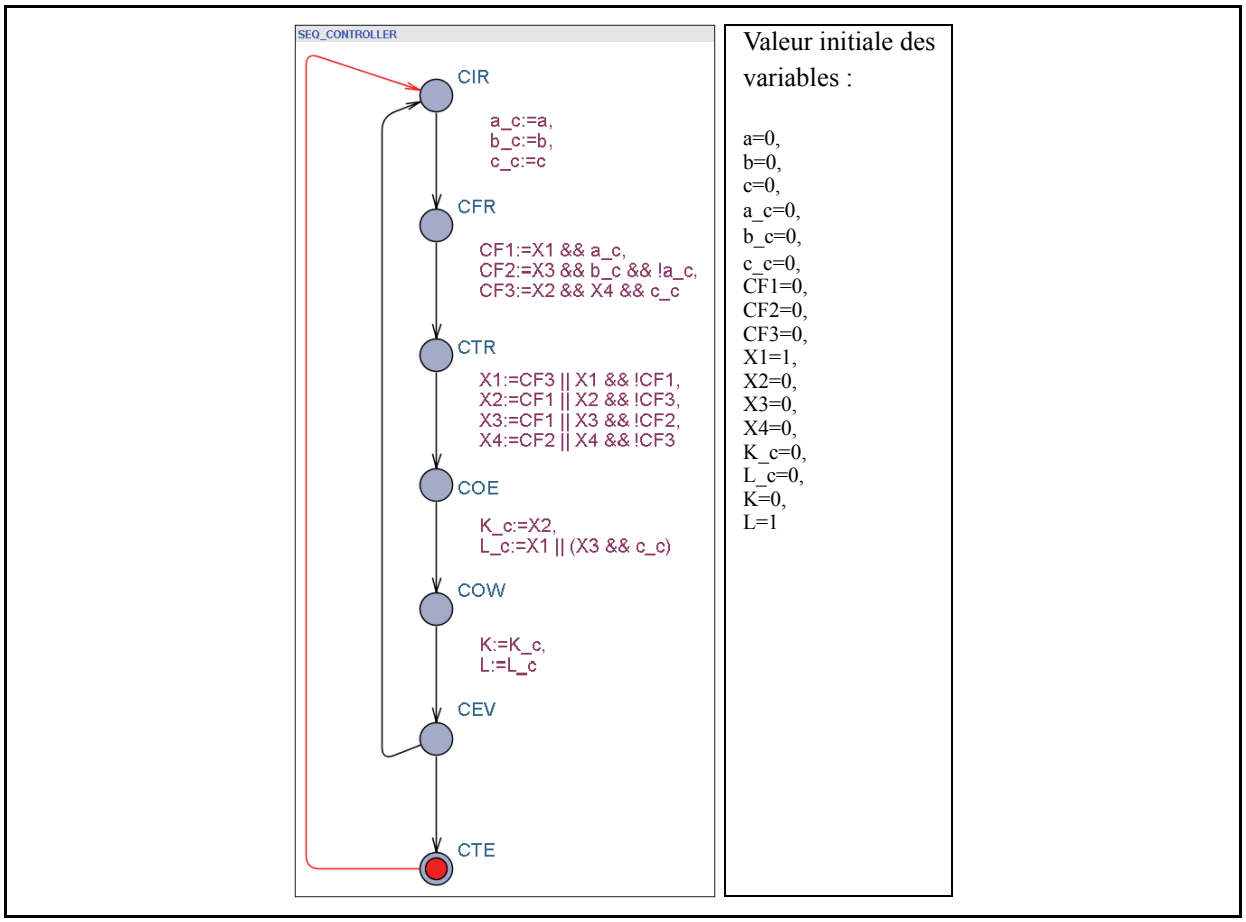


Figure 45: Modèle du contrôleur présenté dans la figure 44.

Cette façon de modéliser le contrôleur est systématique et compacte. On pourra l'appliquer à tous les contrôleurs de caractéristiques similaires dont le programme est exprimé en SFC.

4.2 Modèle des propriétés et de leur séquenceur

Comme nous l'avons indiqué dans les chapitres 1 et 2, les propriétés peuvent être modélisées soit par des expressions en logique temporelle seules, soit avec un automate observateur en complément d'une expression en logique temporelle pour certaines propriétés complexes. Dans ce cas, le modèle de chaque propriété sera considéré comme un module de plus dans notre modèle global. Pour cela, comme pour la modélisation du processus, on a défini un séquenceur pour que tous les modules qui décrivent les propriétés évoluent simultanément. Le séquenceur de propriétés que nous avons défini est composé de trois places (figure 46) :

- la place «SPR» (*Starting PRoperties*) lance l'évolution des modules des propriétés ;
- c'est durant l'activité de la place «DPR» (*Developing PRoperties*) que tous les modules de propriété suivent leur propre évolution ;
- la place «TPR» (*Testing PRoperties*) modélise la fin d'évolution de tous les modèles de propriétés. C'est lorsque cette place est active que doit être faite la preuve de certaines propriétés liées directement aux modèles séquentiels.

Le séquenceur des propriétés aurait pu se limiter à deux places (DPR et TPR), mais pour prendre en compte l'évolution de temporisations logiques qui peuvent être utilisées dans certaines propriétés, comme pour le modèle du processus (chapitre 3), il a été ajouté une place pour que ces temporisations

logiques évoluent dans de bonnes conditions. C'est donc pendant l'activité de la place SPR qu'évolueront les modules des temporisations logiques.

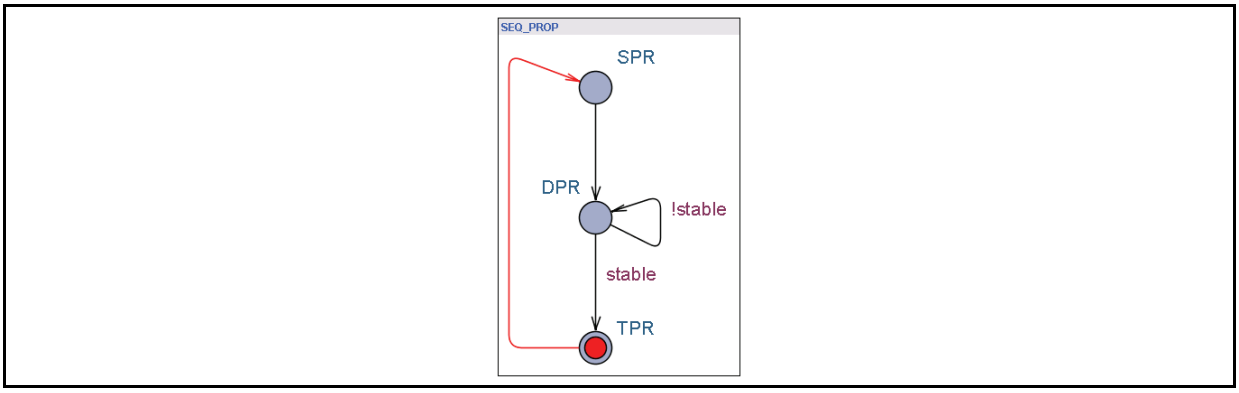


Figure 46: Séquenceur des modules de propriétés.

Comme pour la synchronisation des modules du processus avec leur séquenceur et les modules des temporisations logiques associées, l'intégration des modules des propriétés nécessite des enrichissements. Considérons l'exemple de la figure 47 avec deux automates observateurs d'une propriété (*PR_A*) exprimée avec deux modules (on remarquera que l'évolution du module *PR_A2* dépend du module *PR_A1)*. Les enrichissements à apporter au modèle sont les suivants:

- Pour la recherche de stabilité d'évolution des modules de la propriété (figure 47) :
  - ajouter une variable commune à tous les modules de propriétés «CPRL» (*Change of a PRoperty Location*) qui indique si une évolution a eue lieu dans un des modules de propriétés. Cette variable est assignée la valeur logique «1» à chaque fois qu'une transition entre deux places différentes a été franchie dans un module de propriété ;
  - ajouter une variable par module du modèle des propriétés «FPRT_i» («*Firing of a PRoperty Transition*»), où *i* représente le nom du module auquel est assignée la valeur logique «1» à chaque fois qu'une transition de ce module est franchie.
- Pour la synchronisation avec les modules des temporisations logiques (figure 48):
  - ajouter une variable «FPRT_Ei» («*Firing of a PRoperty Transition of a module Ei*»), où *i* représente le numéro de la temporisation logique associée, à laquelle est assignée la valeur logique «1» à chaque fois qu'est franchie une transition du module de la temporisation *i*. Dans notre exemple, il sera ajouté la variable «FPRT_E1» (figure 48).

Cette façon de modéliser les propriétés, illustrée ici avec un exemple simple, est systématique et peut donc être appliquée à tout ensemble de propriétés séquentielles.

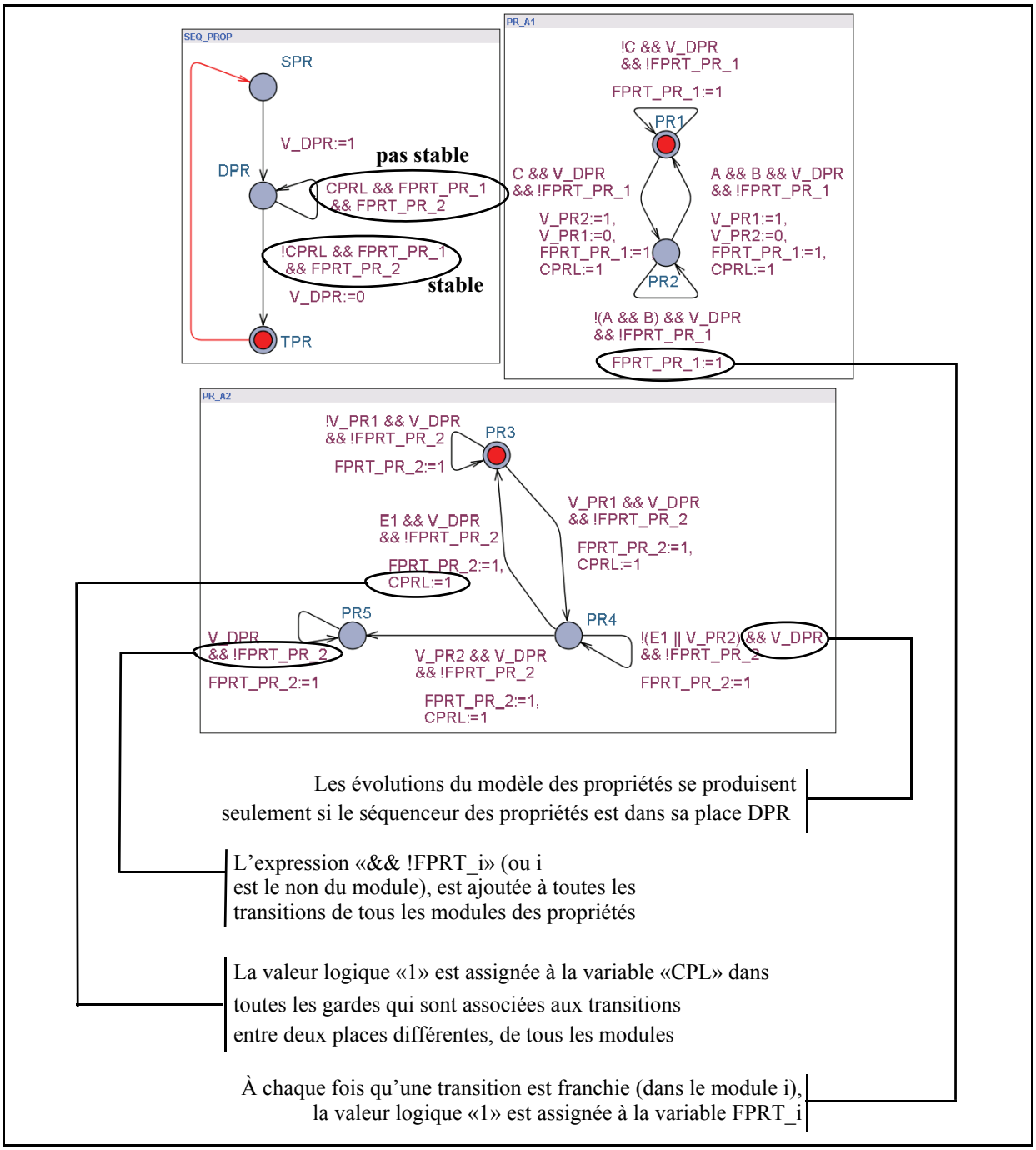


Figure 47: Exemple de modélisation de propriétés séquentielles : variables qui garantissent la stabilité du modèle global des propriétés.

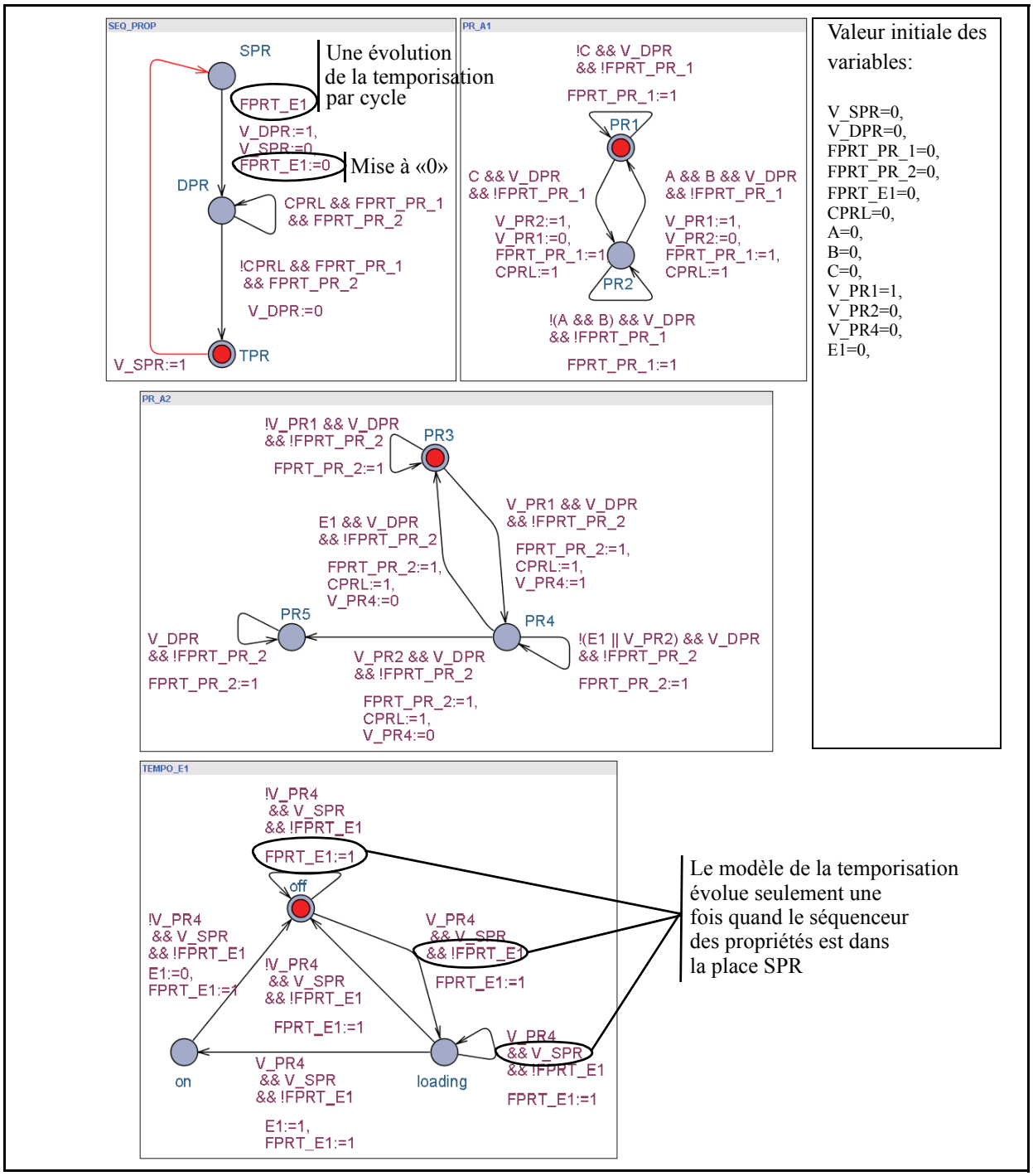


Figure 48: Modèle complet de l'exemple de la figure 47.

4.3 Séquenceur général

Nous avons vu dans le chapitre 3 que l'interaction entre le contrôleur et le processus commandé était traduite par un séquencement qui permettait alternativement la génération d'ordres par le contrôleur et de réactions par le processus. Pour que la preuve de toutes les propriétés puisse être faite correctement, il faut cette fois coordonner l'évolution dynamique du contrôleur, du processus, des propriétés séquentielles et du model-checker. Pour ce faire, nous avons défini un module de synchro-

nisation de l'ensemble de ces modules auquel nous avons donné le non de séquenceur général «*General Sequencer of the system*» («SEQ_GENERAL») (figure 49).

Dans le séquenceur général les cinq places suivantes sont définies:

- la place «START» modélise le début de l'évolution; c'est la place initiale,
- la place «COEV» (*COntroller EVolution*) permet l'évolution du modèle du contrôleur,
- la place «PEAC» (*Properties Evolution After Controller*) autorise l'évolution du modèle des propriétés, après l'évolution du modèle du contrôleur,
- la place «PLEV» (*PLant EVolution*) lance l'évolution du modèle de processus,
- a place «PEAP» (*Properties Evolution After Plant*) modélise l'évolution du modèle des propriétés, après l'évolution du modèle de processus.

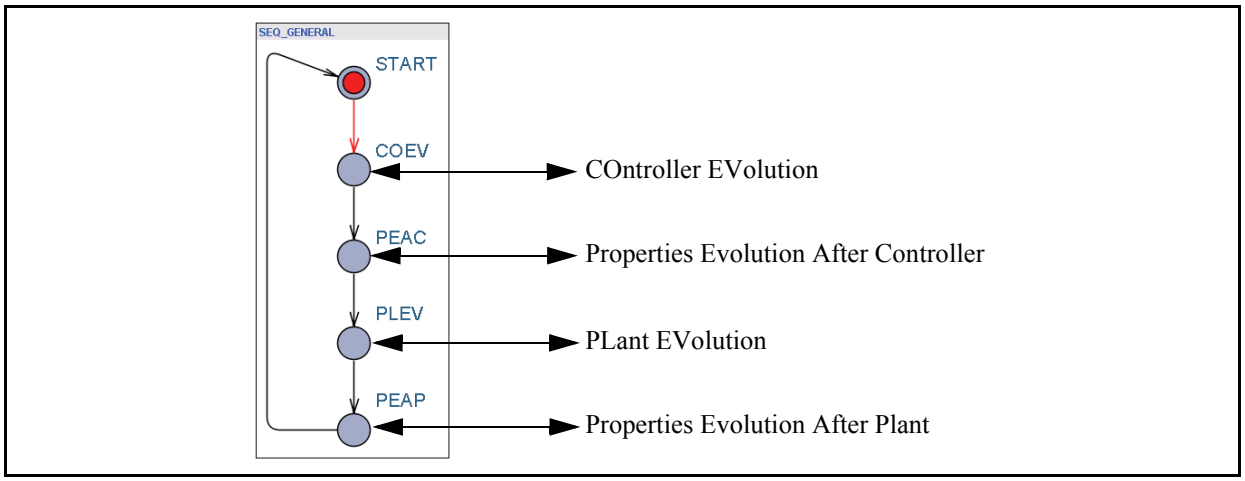


Figure 49: Module du séquenceur général.

Maintenant que nous avons choisi l'ordre d'évolution coordonnée de tous les modèles il faut enrichir les modules des séquenceurs pour garantir leur synchronisation.

- Pour assurer la synchronisation du type appel-réponse avec les autres séquenceurs chaque place du séquenceur général (COEV, PEAC, PLEV, PEAP) sera remplacée par deux places : l'une assure le lancement de l'évolution de chacun des modèles et l'autre correspond à l'évolution de chacun des modèles. Par exemple, la place PLEV est remplacée par les deux places ST_PLEV (*STart PLEV*) et PLEV, où ST_PLEV définit le début de l'évolution du modèle du processus et PLEV correspond à l'évolution elle-même du modèle du processus.
  - À chaque place ST_..... on associe une variable, V_ST_..... qui aura la valeur logique «1» lorsque la place correspondante est active.
- Dans chacun des séquenceurs correspondant aux modèles du contrôleur, du processus et des propriétés,
  - Il faut ajouter une variable aux places qui correspondent à la place de test des propriétés dans chacun des séquenceurs. Cette variable doit avoir la valeur logique «1» pendant l'intervalle de temps où la place est active (V_CTE, pour place de test du séquenceur du contrôleur, V_PTE, pour la place de test du séquenceur du processus et V_PTR pour la place de test du séquenceur des propriétés).

Avec ces variables nous pourrons synchroniser le séquenceur général avec chacun des autres séquenceurs, et ainsi maîtriser correctement l'évolution du modèle global (figure 50).

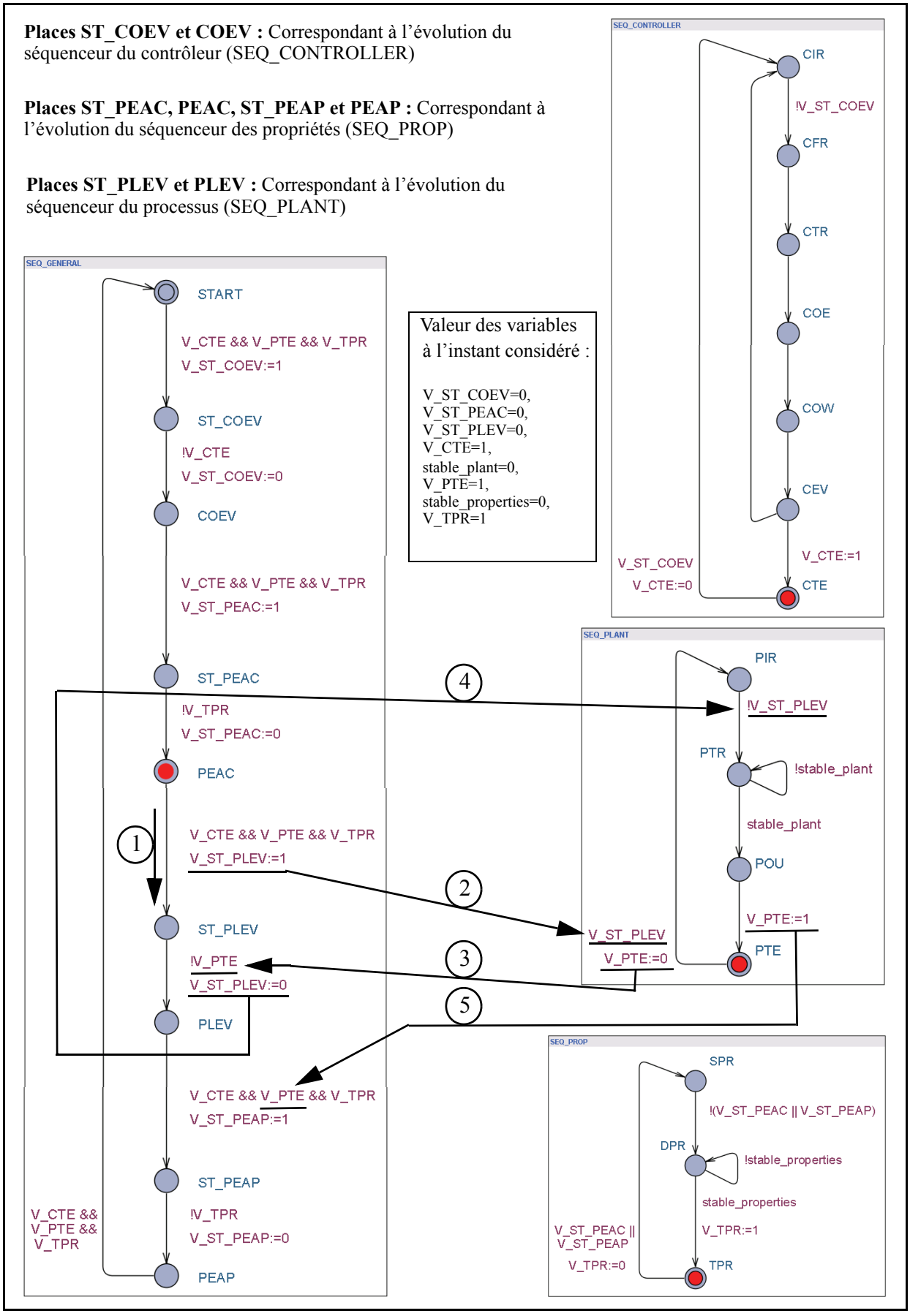


Figure 50: Communication entre séquenceur général et séquenceur du processus.

De manière à illustrer la mise en oeuvre de l'ensemble des modélisations proposées, nous allons maintenant traiter complètement un exemple.

4.4 Présentation de l'exemple support

L'exemple support que nous avons retenu a les caractéristiques suivantes (figure 51) :

- Technologie de puissance électro-pneumatique,
- Diversité des actionneurs (vérin simple/double effet, ventouse, venturi),
- Diversité des pré-actionneurs (distributeurs 3/2, 5/2, monostables, bistables),
- Interdépendance mécanique de chaînes fonctionnelles,
- Diversité de localisation des capteurs (sur actionneur, sur effecteur, sur le produit).

La fonction du système support, qui est de la famille des «pick and place», est de déplacer (grâce à un cycle en «U») des pièces qui se présentent dans trois goulottes d'entrée (détectées par les capteurs «pp1», «pp2» et «pp3») vers une goulotte de sortie. En cas de présence simultanée de pièces dans plusieurs goulottes d'entrée, la priorité est donnée à la pièce se trouvant le plus près de la goulotte de sortie.

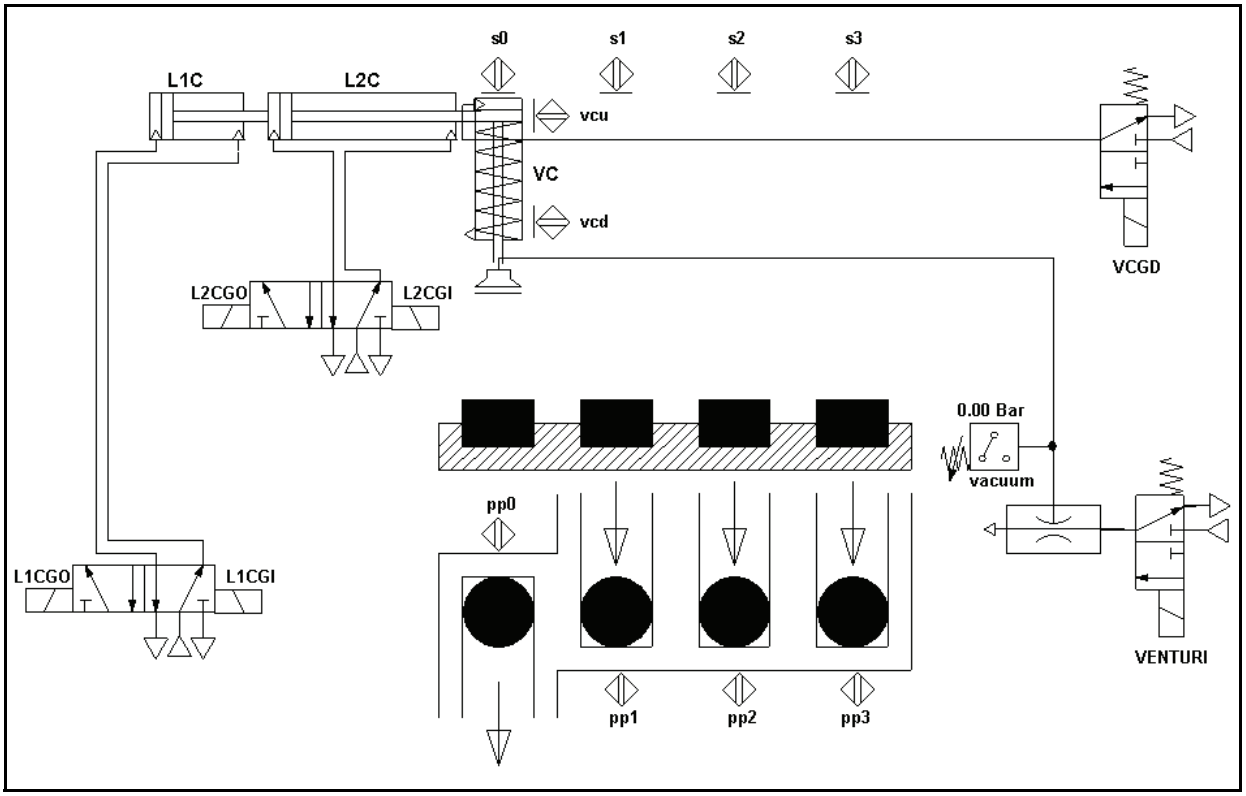


Figure 51: Processus de l'exemple support.

Le processus est composé de deux vérins à double effet (vérins horizontaux) commandés par des distributeurs bistables et d'un vérin à simple effet (vérin vertical) commandé par un distributeur monostable. À l'extrémité de la tige du vérin vertical se trouve une ventouse commandée par un distributeur monostable qui permet la préhension des pièces.

Le vérin horizontal «L1C» (*L1 Cylinder*) a une course moitié moins longue que celle du vérin horizontal «L2C» (*L2 Cylinder*). Les caractéristiques dimensionnelles des vérins horizontaux permettent la prise d'une pièce qui est placée en «pp1» en sortant uniquement la tige du vérin L1C, la prise

d'une pièce qui est placée en «pp2» en sortant uniquement la tige du vérin L2C, la prise d'une pièce qui est placée en «pp3» en sortant les tiges des deux vérins et la dépose d'une pièce dans la goulotte de sortie en rentrant les tiges des deux vérins horizontaux. Les capteurs «s1», «s2» et «s3» sont placés aux endroits qui correspondent à la prise d'une pièce en «pp1», «pp2» et «pp3». Le capteur «s0» est placé à l'endroit où les pièces doivent être déposées dans la goulotte de sortie.

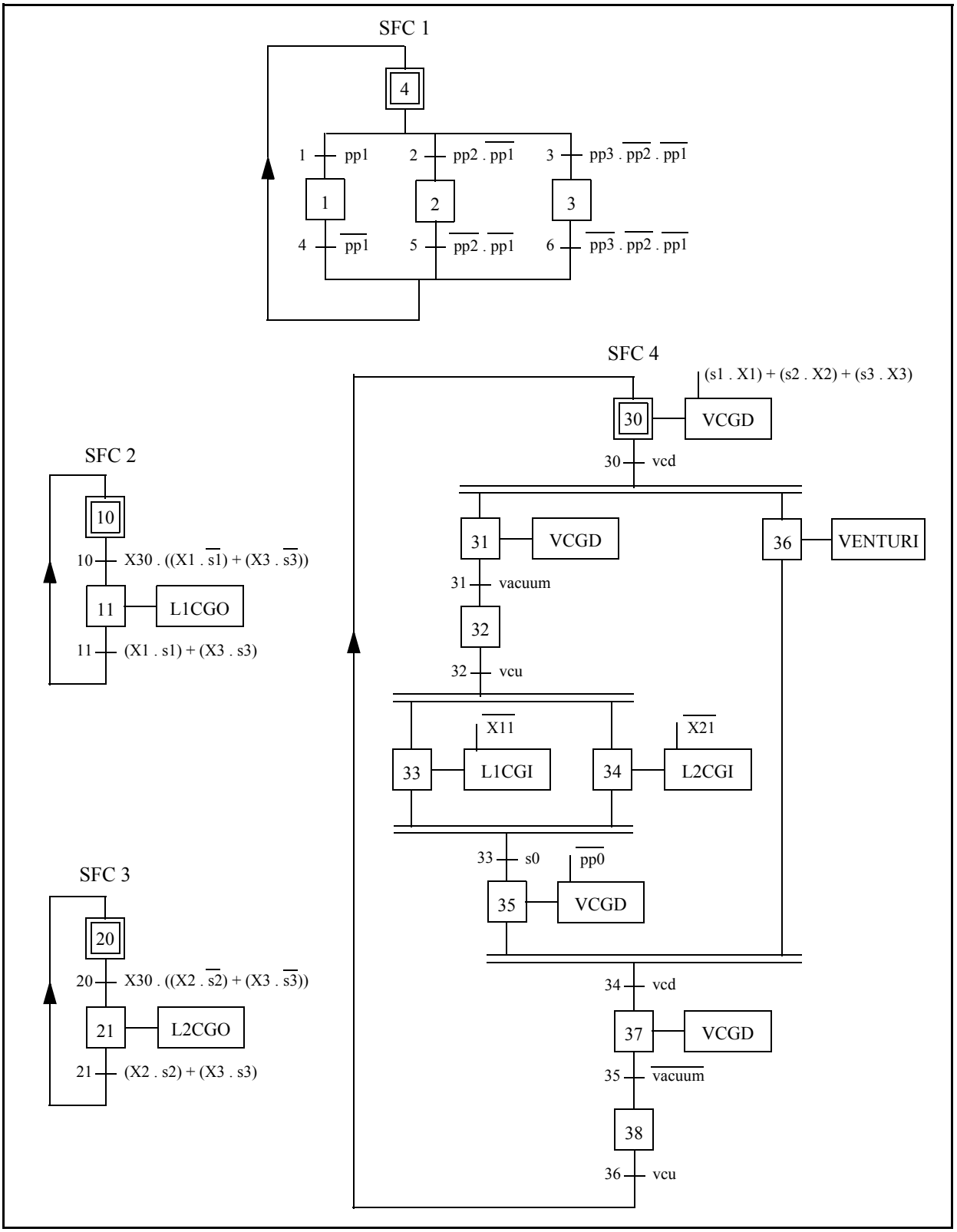


Figure 52: Spécification du contrôleur de l'exemple support.

Le vérin vertical est commandé par un distributeur monostable (ordre «VCGD» (*Vertical Cylinder Go Down*)) et ses positions de fin-de-course sont détectées par les capteurs «vcu» (*vertical cylinder up*) et «vcd» (*vertical cylinder down*). Les vérins horizontaux L1C et L2C sont commandés par des distributeurs bistables et les ordres de commande du distributeur correspondant sont «L1CGO» (*L1C Go Out*) et «L1CGI» (*L1C Go In*). Par analogie, les ordres «L2CGO» et «L2CGI» sont les ordres envoyées au distributeur du vérin L2C pour, respectivement, la sortie et la rentrée de la tige du vérin L2C. Pour la préhension des pièces l'ordre «VENTURI» est envoyé au distributeur associé, et l'aspiration est détectée par le capteur «vacuum».

La spécification du contrôleur de cet étude de cas est présentée figure 52. Le SFC1 assure la mémorisation du cycle à effectuer; les SFC2 et SFC3 gèrent les sorties des tiges des deux vérins horizontaux L1C et L2C et le SFC4 coordonne le mouvement d'un cycle en «U» de prise-dépose.

Les propriétés que l'on souhaite vérifier sur ce contrôleur sont les suivantes :

- PR_1 : «Les ordres envoyés au distributeur qui commande le vérin L1C ne doivent jamais être émis en même temps»; «Les ordres envoyés au distributeur qui commande le vérin L2C ne doivent jamais être émis en même temps».
- PR_2 : «Si l'ordre de descendre le vérin vertical est présent, alors on ne doit pas avoir l'émission d'ordres de commande relatifs aux mouvements des vérins horizontaux».
- PR_3 : «S'il y a émission d'un ordre de commande correspondant aux mouvements des vérins horizontaux alors le capteur «vcu» doit être à «1»».
- PR_4 : «Après avoir été saisie, une pièce ne peut être lâchée qu'au droit de sa position de dépose».
- PR_5 : «Il n'y a de mouvements horizontaux que si le capteur «vcu» est à «1»».
- PR_6 : «Le modèle du contrôleur est toujours vivant».
- PR_7 : «Si une pièce est détectée par pp1, pp2 ou pp3, alors on aura forcement, dans le futur, la sortie d'un des vérins horizontaux».
- PR_8 : «Si une pièce est détectée par pp1, pp2 ou pp3, alors on aura forcement, dans le futur, l'aspiration de la pièce».
- PR_9 : «Quand le vérin vertical descend, tous les autres vérins sont et restent en fin de course».

Tableau 1: Caractéristiques de propriétés à prouver.

	Type de propriété		Variables utilisées dans l'expression de la propriété		
	Sûreté	Vivacité	Variables contrôlables	Variables observables	Variables processus
PR_1	X		X		
PR_2	X		X		
PR_3	X		X	X	
PR_4	X			X	
PR_5	X			X	X
PR_6		X	X		
PR_7		X		X	X
PR_8		X		X	
PR_9	X				X

Ces propriétés ont été choisies pour être représentatives des différents types de propriétés (on retiendra comme [ALPERN et al. 1985] et [MANNA et al. 1995], les propriétés de *sûreté* et les proprié-

tés de *vivacité*) et de différentes syntaxes pour les exprimer. Pour en proposer une classification, nous avons distingué trois types de variables utilisées dans l'expression des propriétés :

- Les *variables contrôlables* par le contrôleur (variables d'état et sorties du modèle du contrôleur),
- Les *variables observables* par le contrôleur (entrées du contrôleur).
- Les *variables processus* (variables d'état du modèle du processus).

4.5 Modèle global du système

Dans cette section on présente le modèle de l'exemple support en détaillant les modules les uns après les autres. Le lecteur trouvera dans l'annexe 1 le modèle complet qui, pour des raisons de taille, n'est pas présenté in extenso dans ce chapitre.

4.5.1 Modèle du contrôleur

Comme illustré dans la section "Modèle du contrôleur et de son séquenceur", page 57, le modèle du contrôleur et de son séquenceur est élaboré sous une forme compacte par un seul module, où à chaque évolution du modèle sont calculées les variables correspondant au programme SFC et également les variables partagées pour la communication entre ce module et les autres modules du modèle global.

Le modèle du contrôleur (qui avait été spécifié figure 52) est présenté dans la figure 53. La place initiale est la place «CTE» et l'évolution du modèle du contrôleur commence lorsque la variable «V_ST_COEV» est à la valeur logique «1». Cette variable correspond à la communication entre le module du contrôleur et le module du séquenceur général. La variable «V_CTE» de ce module sert également à la communication avec les autres modules du modèle global.

Les équations de traduction algébrique du SFC présenté dans la figure 52 sont les suivantes :

- Pour les conditions de franchissement (conformément à l'équation (6)):

$$CF(1)(t) = X4(t) \wedge pp1$$
$$CF(2)(t) = X4(t) \wedge pp2 \wedge \neg pp1$$
$$CF(3)(t) = X4(t) \wedge pp3 \wedge \neg pp2 \wedge \neg pp1$$
$$CF(4)(t) = X1(t) \wedge \neg pp1$$
$$CF(5)(t) = X2(t) \wedge \neg pp2 \wedge \neg pp1$$
$$CF(6)(t) = X3(t) \wedge \neg pp3 \wedge \neg pp2 \wedge \neg pp1$$
$$CF(10)(t) = X10(t) \wedge X30(t) \wedge ((X1(t) \wedge \neg s1) \vee (X3(t) \wedge \neg s3))$$
$$CF(11)(t) = X11(t) \wedge ((X1(t) \wedge s1) \vee (X3(t) \wedge s3))$$
$$CF(20)(t) = X20(t) \wedge X30(t) \wedge ((X2(t) \wedge \neg s2) \vee (X3(t) \wedge \neg s3))$$
$$CF(21)(t) = X21(t) \wedge ((X2(t) \wedge s2) \vee (X3(t) \wedge s3))$$
$$CF(30)(t) = X30(t) \wedge vcd$$
$$CF(31)(t) = X31(t) \wedge vacuum$$
$$CF(32)(t) = X32(t) \wedge vcu$$
$$CF(33)(t) = X33(t) \wedge X34(t) \wedge s0$$
$$CF(34)(t) = X35(t) \wedge X36(t) \wedge vcd$$
$$CF(35)(t) = X37(t) \wedge \neg vacuum$$
$$CF(36)(t) = X38(t) \wedge vcu$$

- Pour les états d'étapes (conformément à l'équation (2)):

$$X4(t) = CF(4)(t) \vee CF(5)(t) \vee CF(6)(t) \vee X4(t-1) \wedge \neg(CF(1)(t) \vee CF(2)(t) \vee CF(3)(t))$$
$$X1(t) = CF(1)(t) \vee X1(t-1) \wedge \neg CF(4)(t)$$
$$X2(t) = CF(2)(t) \vee X2(t-1) \wedge \neg CF(5)(t)$$
$$X3(t) = CF(3)(t) \vee X3(t-1) \wedge \neg CF(6)(t)$$
$$X10(t) = CF(11)(t) \vee X10(t-1) \wedge \neg CF(10)(t)$$
$$X11(t) = CF(10)(t) \vee X11(t-1) \wedge \neg CF(11)(t)$$
$$X20(t) = CF(21)(t) \vee X20(t-1) \wedge \neg CF(20)(t)$$
$$X21(t) = CF(20)(t) \vee X21(t-1) \wedge \neg CF(21)(t)$$
$$X30(t) = CF(36)(t) \vee X30(t-1) \wedge \neg CF(30)(t)$$
$$X31(t) = CF(30)(t) \vee X31(t-1) \wedge \neg CF(31)(t)$$
$$X32(t) = CF(31)(t) \vee X32(t-1) \wedge \neg CF(32)(t)$$
$$X33(t) = CF(32)(t) \vee X33(t-1) \wedge \neg CF(33)(t)$$
$$X34(t) = CF(32)(t) \vee X34(t-1) \wedge \neg CF(33)(t)$$
$$X35(t) = CF(33)(t) \vee X35(t-1) \wedge \neg CF(34)(t)$$
$$X36(t) = CF(30)(t) \vee X36(t-1) \wedge \neg CF(34)(t)$$
$$X37(t) = CF(34)(t) \vee X37(t-1) \wedge \neg CF(35)(t)$$
$$X38(t) = CF(35)(t) \vee X38(t-1) \wedge \neg CF(36)(t)$$

- Pour les sorties (conformément à l'équation (7)):

$$VENTURI(t) = X36(t)$$
$$VCGD(t) = (X30(t) \wedge ((X1(t) \wedge s1(t)) \vee (X2(t) \wedge s2(t)) \vee (X3(t) \wedge s3(t)))) \vee X31(t) \vee (X35(t) \wedge pp0(t)) \vee X37(t)$$
$$L1CGI(t) = (X33(t) \wedge \neg X11(t))$$
$$L2CGI(t) = (X34(t) \wedge \neg X21(t))$$
$$L1CGO(t) = X11(t)$$
$$L2CGO(t) = X21(t)$$

En intégrant ces expressions algébriques dans le séquenceur du contrôleur, nous obtenons le modèle du contrôleur de la figure 53.

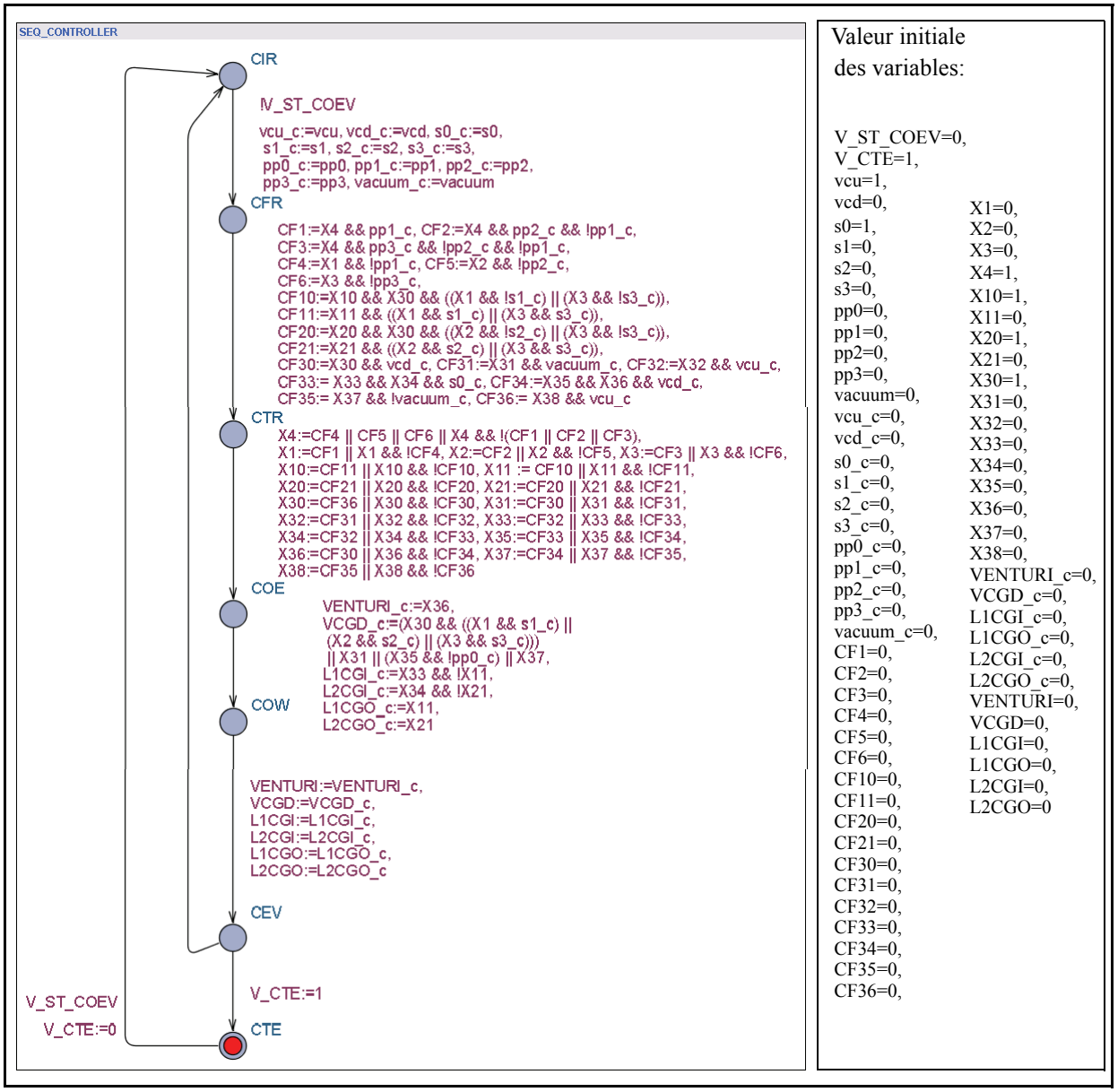


Figure 53: Modèle du contrôleur spécifié figure 52.

4.5.2 Modèle du processus

Comme indiqué précédemment dans le chapitre 3, la modélisation du processus est une tâche qui nécessite l'expertise du concepteur. En prenant en compte les niveaux de granularité (de structuration et de comportement) mentionnés dans le chapitre précédent, nous opterons par le découpage du processus en modules conformément à la figure 54.

Les modules suivants ont donc été retenus :

- **VC** : Composé du vérin vertical «VC», de son distributeur et des deux capteurs fin de course «vcu» et «vcd»;
- **L1C** : Composé du vérin horizontal «L1C» et de son distributeur;
- **L2C** : Composé du vérin horizontal «L2C» et de son distributeur;
- **sensor_s0** : Composé du capteur «s0» qui dépend des chaînes fonctionnelles des deux vérins horizontaux;
- **sensor_s1** : Composé du capteur «s1» qui dépend des chaînes fonctionnelles des deux

vérins horizontaux;

- **sensor_s2** : Composé du capteur «s2» qui dépend des chaînes fonctionnelles des deux vérins horizontaux;
- **sensor_s3** : Composé du capteur «s3» qui dépend des chaînes fonctionnelles des deux vérins horizontaux;
- **ZONE_V** : Composé de la ventouse, de son capteur «vacuum» et du comportement local du produit;
- **ZONE_0** : Composé du capteur «pp0» et du comportement local du produit;
- **ZONE_1** : Composé du capteur «pp1» et du comportement local du produit;
- **ZONE_2** : Composé du capteur «pp2» et du comportement local du produit;
- **ZONE_3** : Composé du capteur «pp3» et du comportement local du produit;

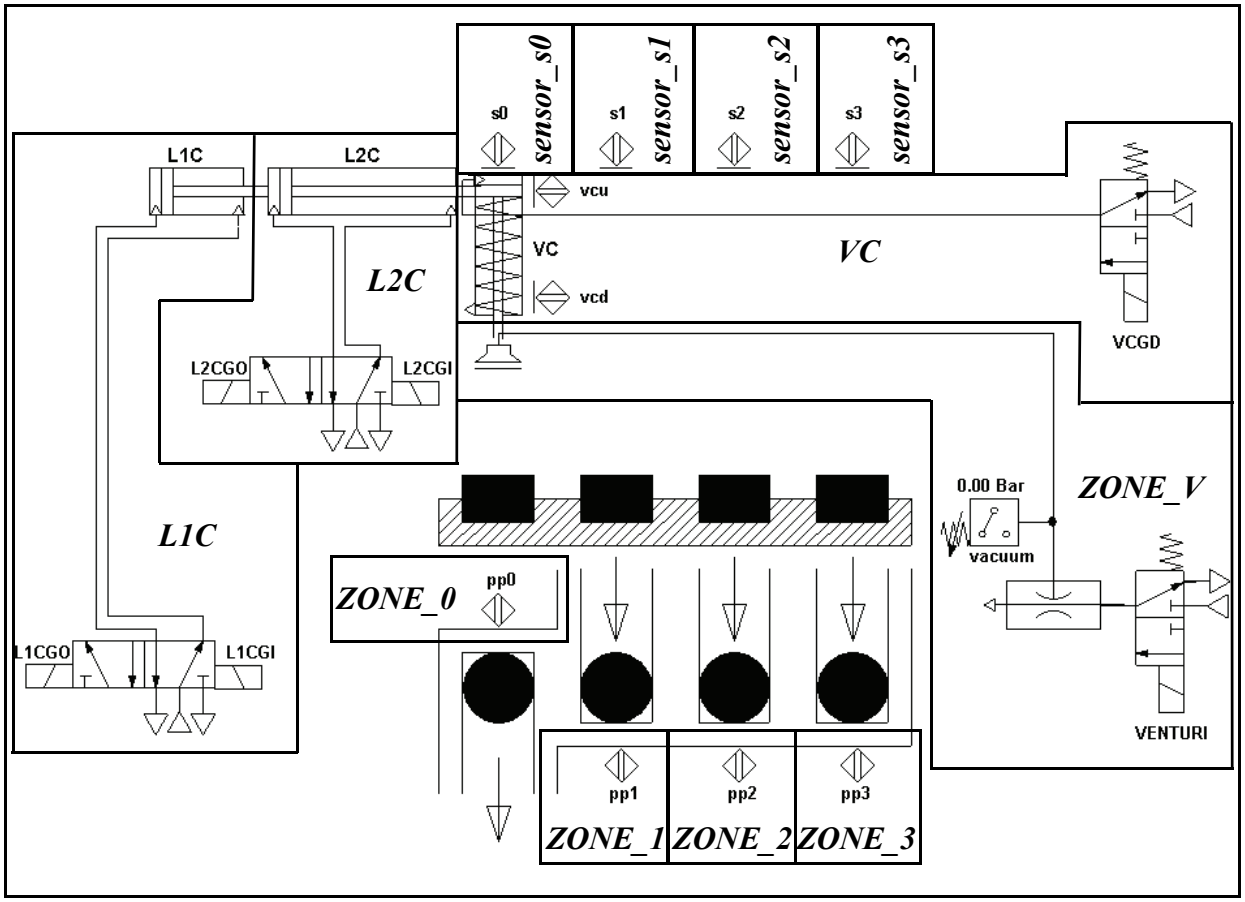


Figure 54: Modules du processus de l'exemple.

4.5.2.1 Module VC

Le module VC est le modèle d'une chaîne fonctionnelle composée d'un vérin simple effet (vérin vertical), commandé par un distributeur monostable, et de deux capteurs de fin de course qui indiquent les deux positions extrêmes de la tige du vérin.

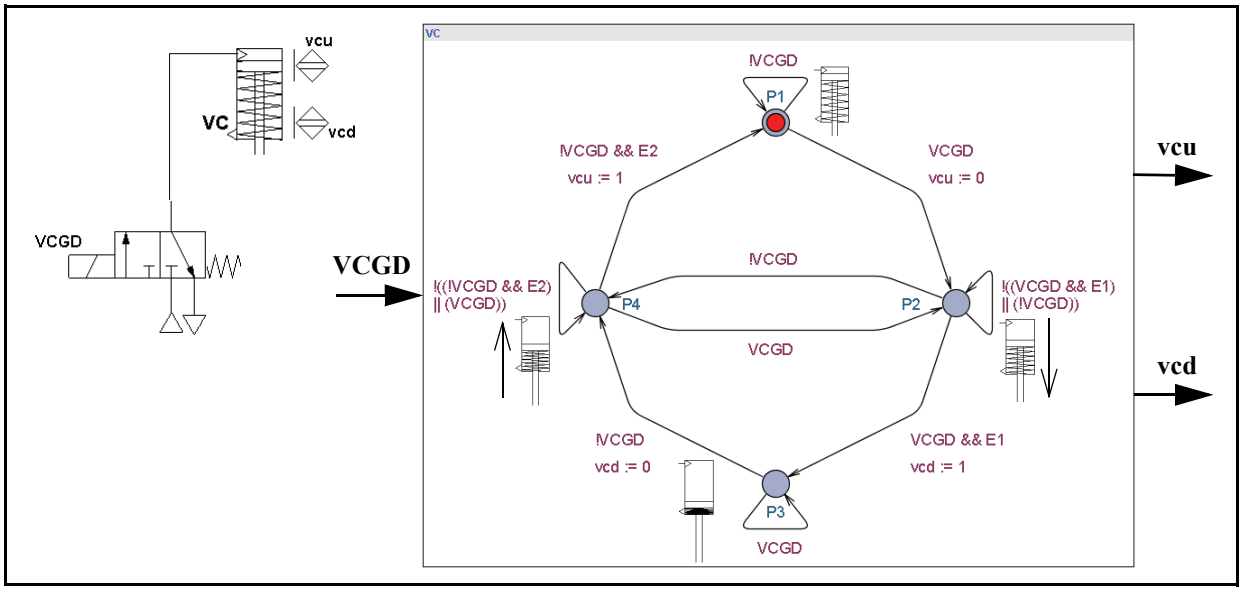


Figure 55: Modèle du module VC (chaîne fonctionnelle pneumatique du vérin vertical).

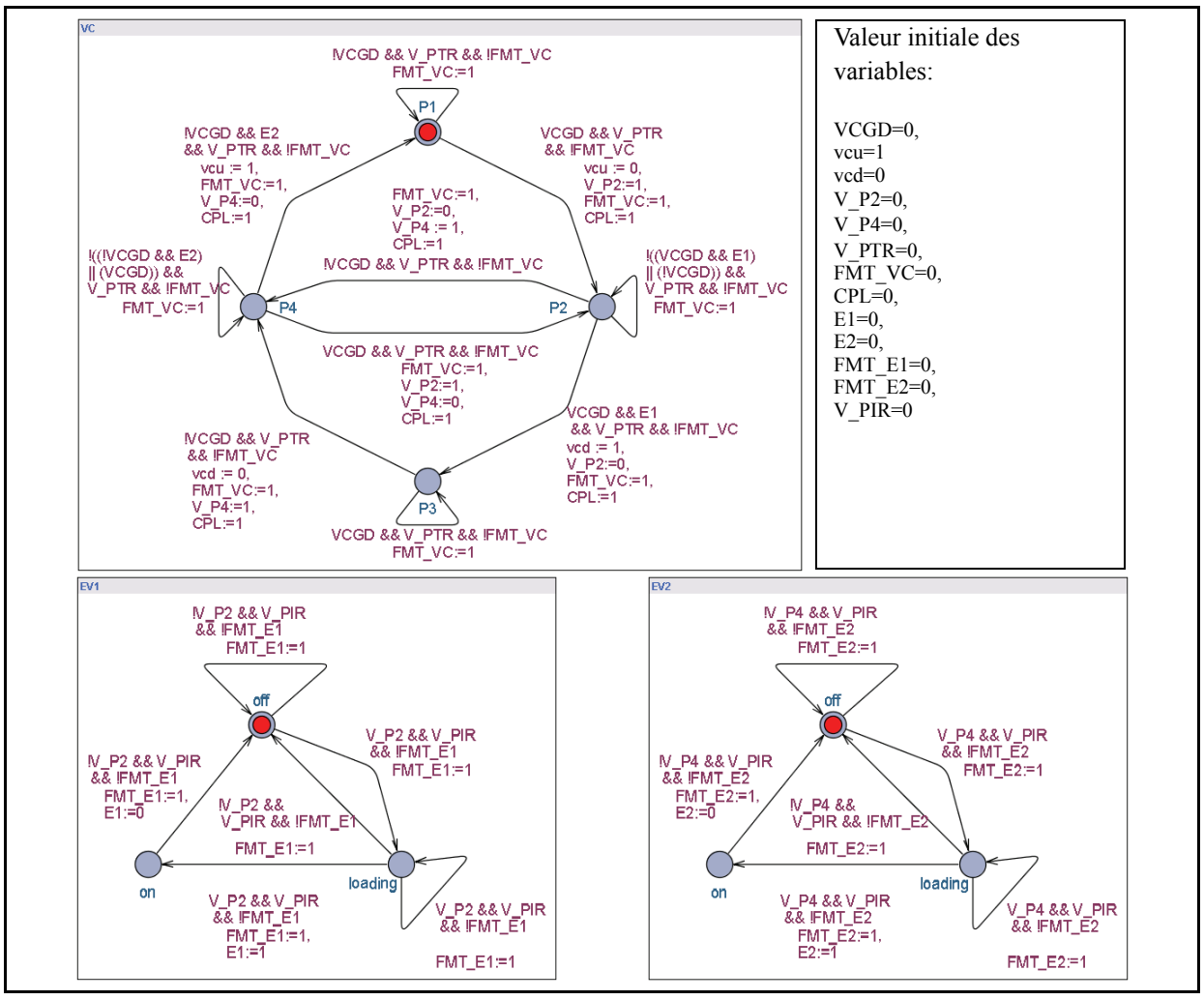


Figure 56: Modèle complet du module VC (chaîne fonctionnelle pneumatique correspondant au vérin vertical).

Selon le point de vue du processus, ce modèle produit les sorties «vcu» et «vcd» à partir de l'entrée «VCGD».

- P1 : Place qui modélise la tige du vérin VC en position rentrée;
- P2 : Place qui modélise la tige du vérin VC en train de sortir;
- P3 : Place qui modélise la tige du vérin VC en position sortie;
- P4 : Place qui modélise la tige du vérin VC en train de rentrer.

Les variables ajoutées au modèle sont :

- E1 : Temporisation logique modélisant le temps de sortie de la tige du vérin VC;
- E2 : Temporisation logique modélisant le temps de rentrée de la tige du vérin VC;
- V_P2 qui déclenche la temporisation logique E1;
- V_P4 qui déclenche la temporisation logique E2;

Pour l'obtention du modèle complet on a dû ajouter :

- V_PTR, CPL et FMT_VC pour la synchronisation entre VC et le séquenceur du processus;
- V_PIR, FMT_E1 et FMT_E2 pour la synchronisation entre les modules des deux temporisations logiques et le séquenceur du processus, comme présenté figure 56.

4.5.2.2 Module L1C

Le module L1C est le modèle d'une chaîne fonctionnelle composée d'un vérin double effet (vérin horizontal L1C), commandé par un distributeur bistable. À la différence du modèle précédent (VC), ce modèle ne produit pas directement d'information capteur. Seules les activités de places sont utilisées par les autres modèles.

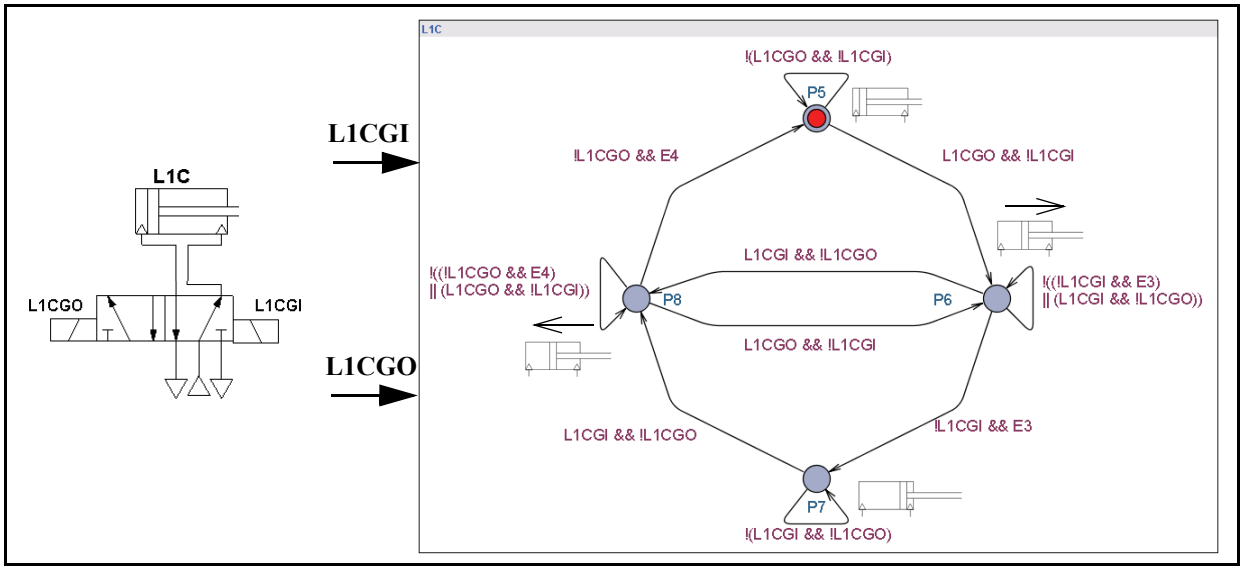


Figure 57: Modèle simplifié du module L1C (chaîne fonctionnelle pneumatique correspondant au vérin vertical L1C).

Du point de vue du processus, ses entrées sont L1CGI et L1CGO.

- P5 : Place qui modélise la tige du vérin L1C en position rentrée;
- P6 : Place qui modélise la tige du vérin L1C en train de sortir;
- P7 : Place qui modélise la tige du vérin L1C en position sortie;
- P8 : Place qui modélise la tige du vérin L1C en train de rentrer.

Les variables ajoutées au modèle sont :

- E3 : Temporisation logique modélisant le temps de sortie de la tige du vérin L1C;
- E4 : Temporisation logique modélisant le temps de rentrée de la tige du vérin L1C;

- V_P6 qui déclenche la temporisation logique E3;
- V_P8 qui déclenche la temporisation logique E4;

Pour l'obtention du modèle complet on a dû ajouter :

- V_PTR, CPL et FMT_L1C pour la synchronisation entre L1C et le séquenceur du processus;
- V_PIR, FMT_E3 et FMT_E4 pour la synchronisation entre les modules des deux temporisations logiques et le séquenceur du processus, comme présenté figure 58.

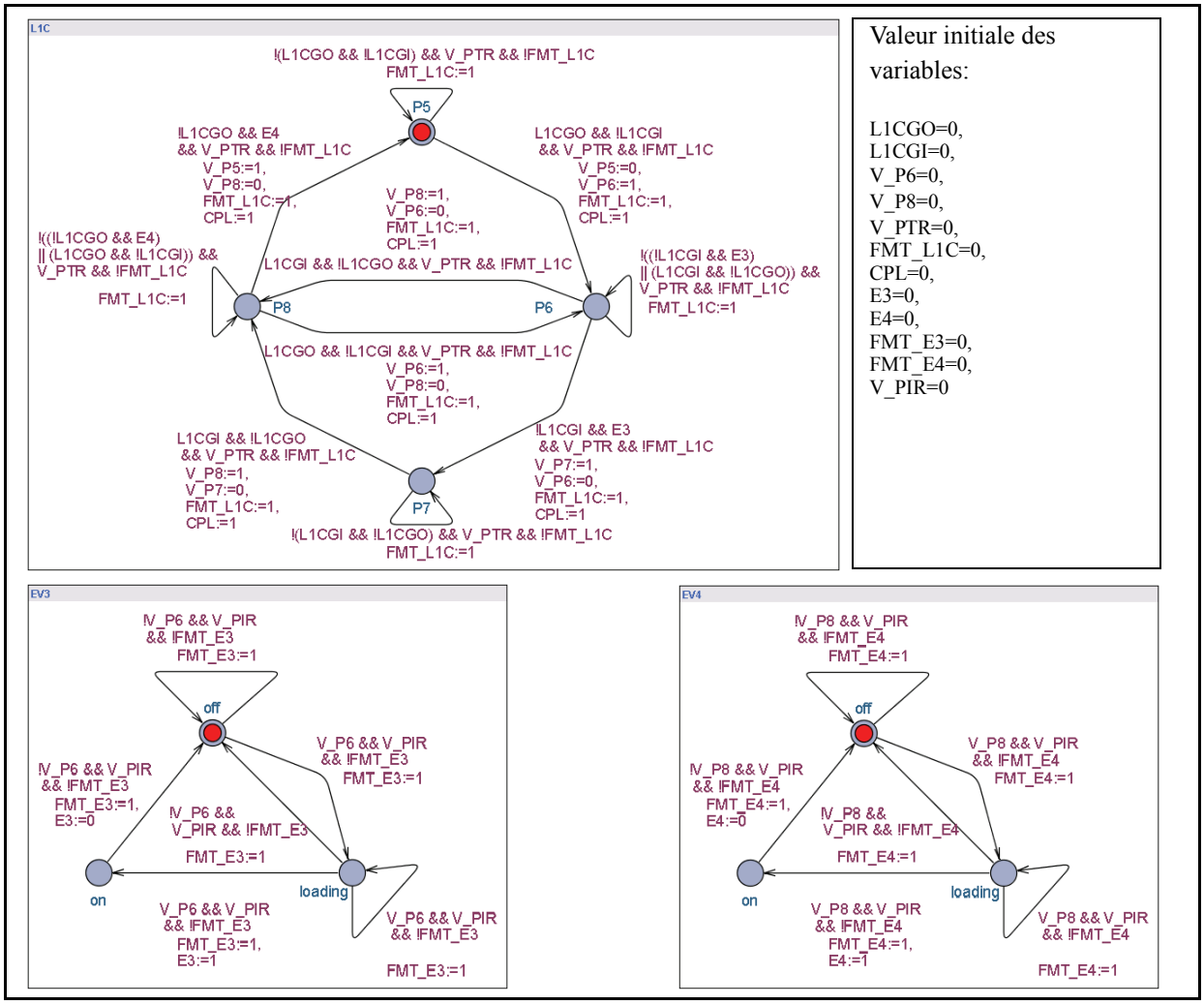


Figure 58: Modèle global final de la chaîne fonctionnelle pneumatique correspondant au vérin horizontal L1C.

Le modèle du module L2C (de la chaîne fonctionnelle correspondante au vérin horizontal L2C) est semblable au modèle présenté pour le module L1C. Il est présenté dans l'annexe 1.

4.5.2.3 Module sensor_s1

Le comportement de tout capteur si («i» vaut 0, 1, 2, ou 3) ne dépend pas directement du comportement d'un seul module, mais de deux chaînes fonctionnelles (modules L1C et L2C). Comme il a été expliqué dans le chapitre 3, nous construirons donc un modèle pour chaque capteur, indépendant des modèles de chacune des chaînes fonctionnelles. Nous allons présenter la construction d'un tel module pour le capteur s1.

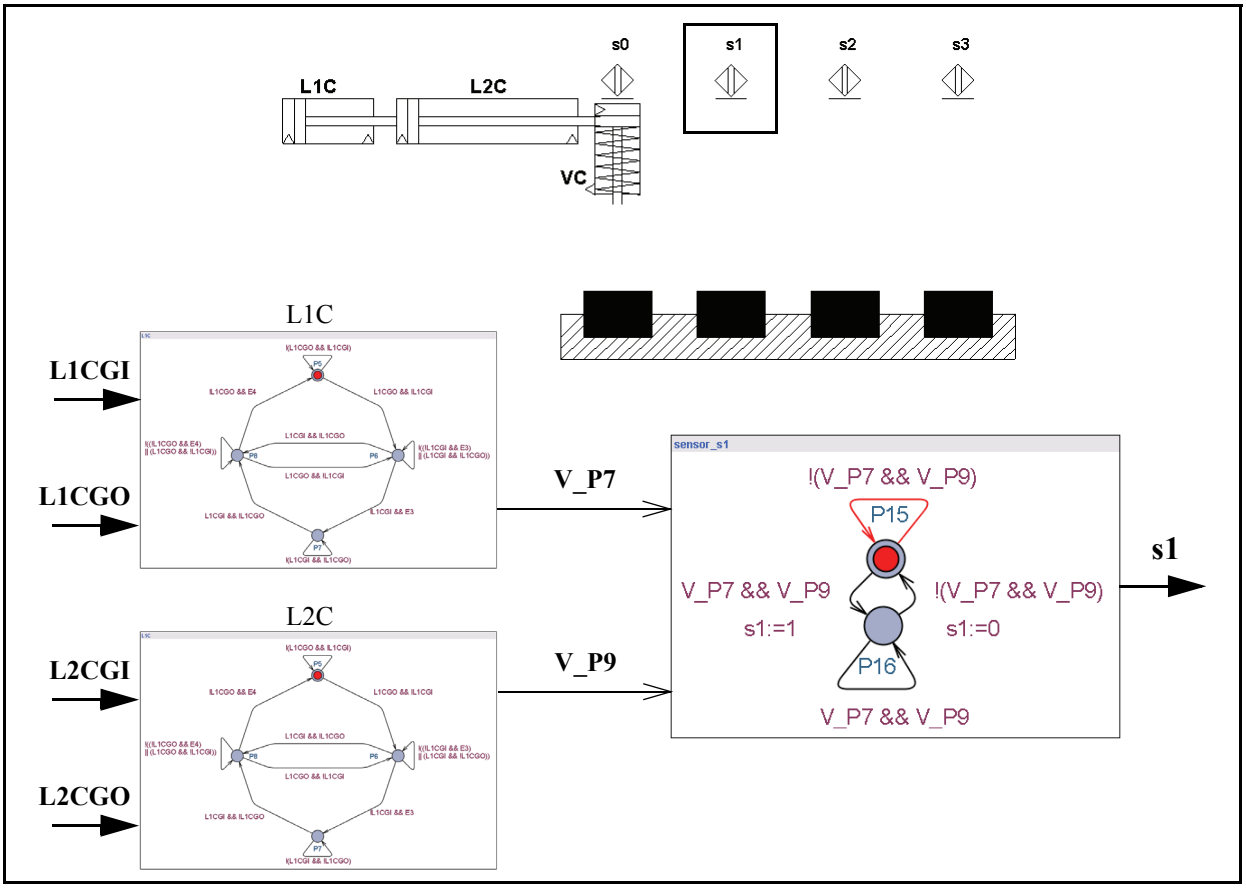


Figure 59: Modèle de comportement du capteur s1 (module sensor_s1).

Du point de vue du processus, les entrées de ce module sont «V_P7» et «V_P9». Ces variables ne sont pas des entrées du processus, mais des variables de coordination entre modules. La sortie de ce module est «s1».

- P15 : Place qui modélise le capteur inactif;
- P16 : Place qui modélise le capteur actif;

Pour l'obtention du modèle complet on a dû ajouter les variables :

- V_PTR, CPL et FMT_s1p pour la synchronisation entre le module sensor_s1 et le séquenceur du processus; figure 60.

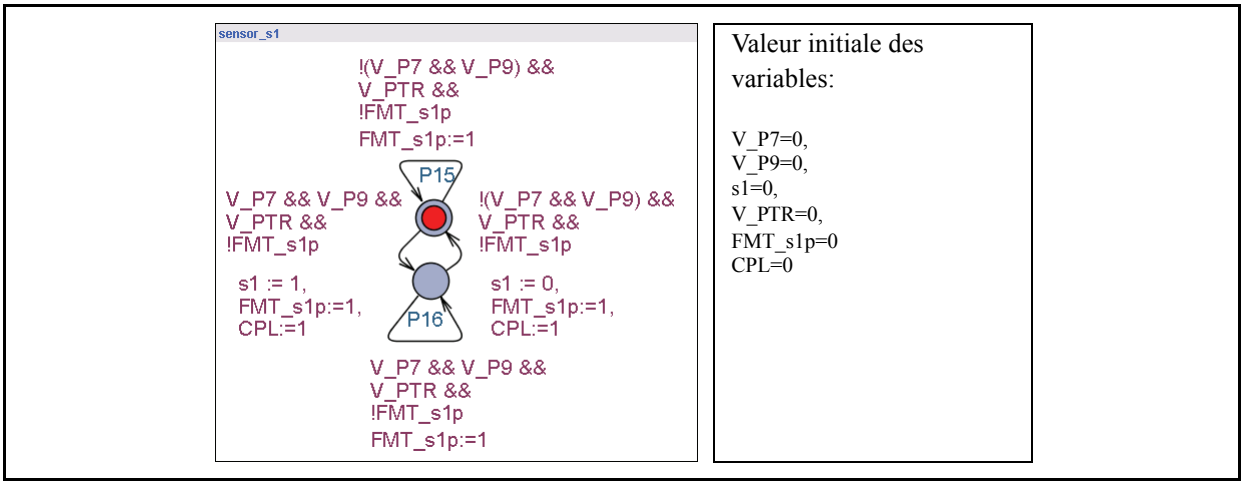


Figure 60: Modèle complet du module sensor_s1.

Le modèle de chacun des autres capteurs si (sensor_s0, sensor_s2 et sensor_s3) est similaire à celui présenté pour le module sensor_s1. Ces modèles sont présentés dans l'annexe 1.

4.5.2.4 Module ZONE_2

Dans le système, nous avons trois goulottes pour l'arrivée de pièces détectées par pp1, pp2, et/ou pp3. Le comportement de chacun de ces capteurs dépend de l'arrivée physique (on pourrait également parler de comportement cinématique) des produits. Nous avons donc choisi de ne pas modéliser indépendamment le capteur ou le produit mais l'ensemble des deux. De ce fait, nous construisons quatre modèles que nous avons nommé «ZONE_i» («i» est l'indice de chaque capteur). Analysons, en détail, ce qui se passe, par exemple dans la ZONE_2, dont le modèle simplifié est présenté figure 61.

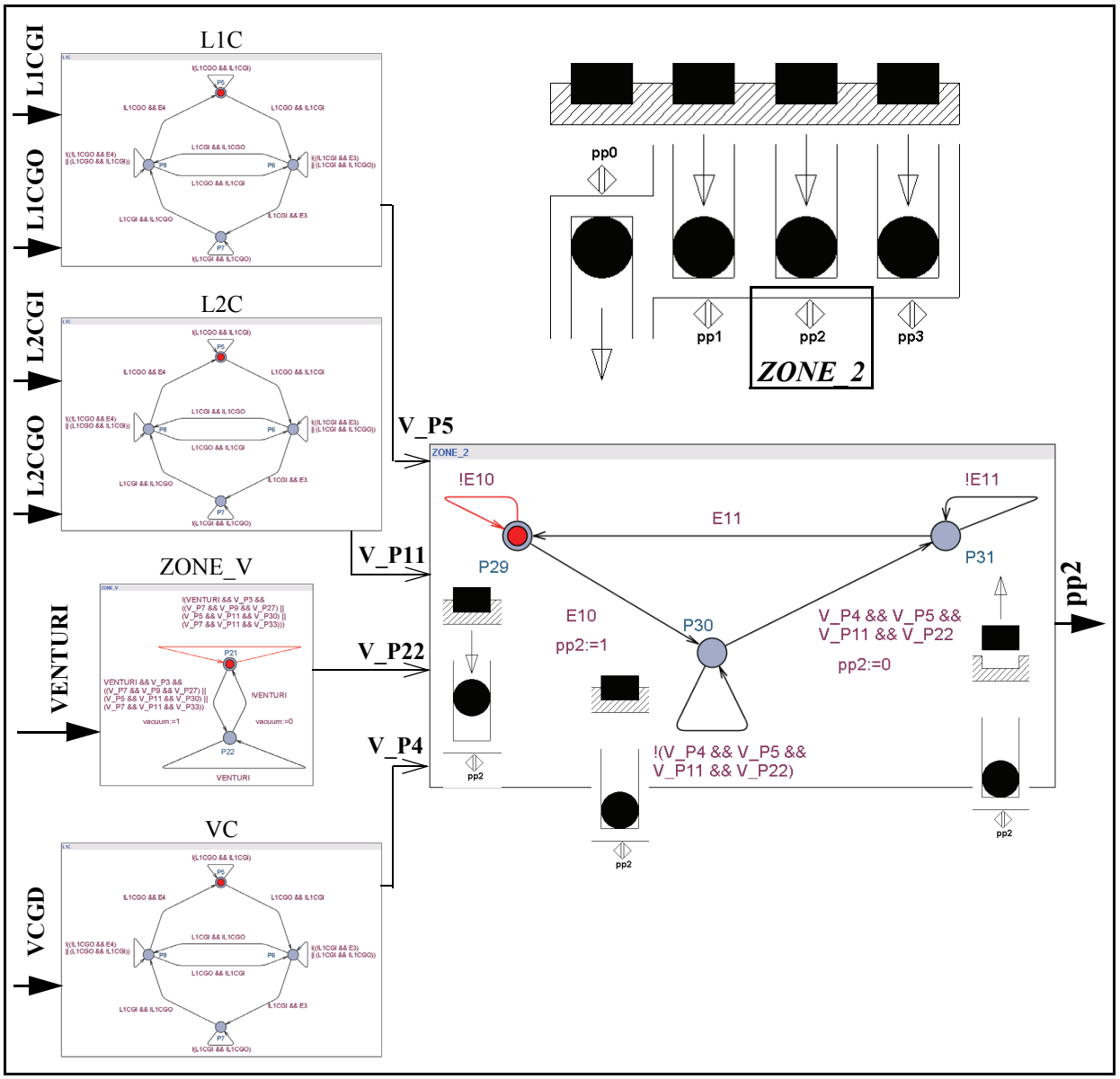


Figure 61: Modèle simplifié du module ZONE_2. dépend des modules L1C, L2C, ZONE_V et VC).

Du point de vue du processus, les entrées du module ZONE_2 sont des variables générées par d'autres modules du processus (V_P5, V_P11, V_P22, V_P4) et sa sortie est «pp2».

- P29 : Place qui modélise l'arrivée d'une pièce dans la goulotte du capteur pp2;
- P30 : Place qui modélise la présence d'une pièce détectée par le capteur pp2;

- P31 : Place qui modélise la sortie d'une pièce de la goulotte du capteur pp2;

Les variables ajoutées au modèle sont :

- E10 : Temporisation logique modélisant le temps d'arrivée d'une pièce dans la zone de détection de pp2;
- E11 : Temporisation logique modélisant le temps entre la sortie et l'arrivée d'une pièce;
- V_P29 qui déclenche la temporisation logique E10;
- V_P31 qui déclenche la temporisation logique E11;

Pour l'obtention du modèle complet on a dû ajouter :

- V_PTR, CPL et FMT_Z2 pour la synchronisation entre le module ZONE_2 et le séquenceur du processus;
- V_PIR, FMT_E10 et FMT_E11 pour la synchronisation entre les modules des deux temporisations logiques et le séquenceur du processus, comme présenté figure 62.

Pour la communication avec le module ZONE_V on a dû ajouter:

- V_P30 qui indique quand une pièce est présente pour être prise (voir "Module ZONE_V", page 81).

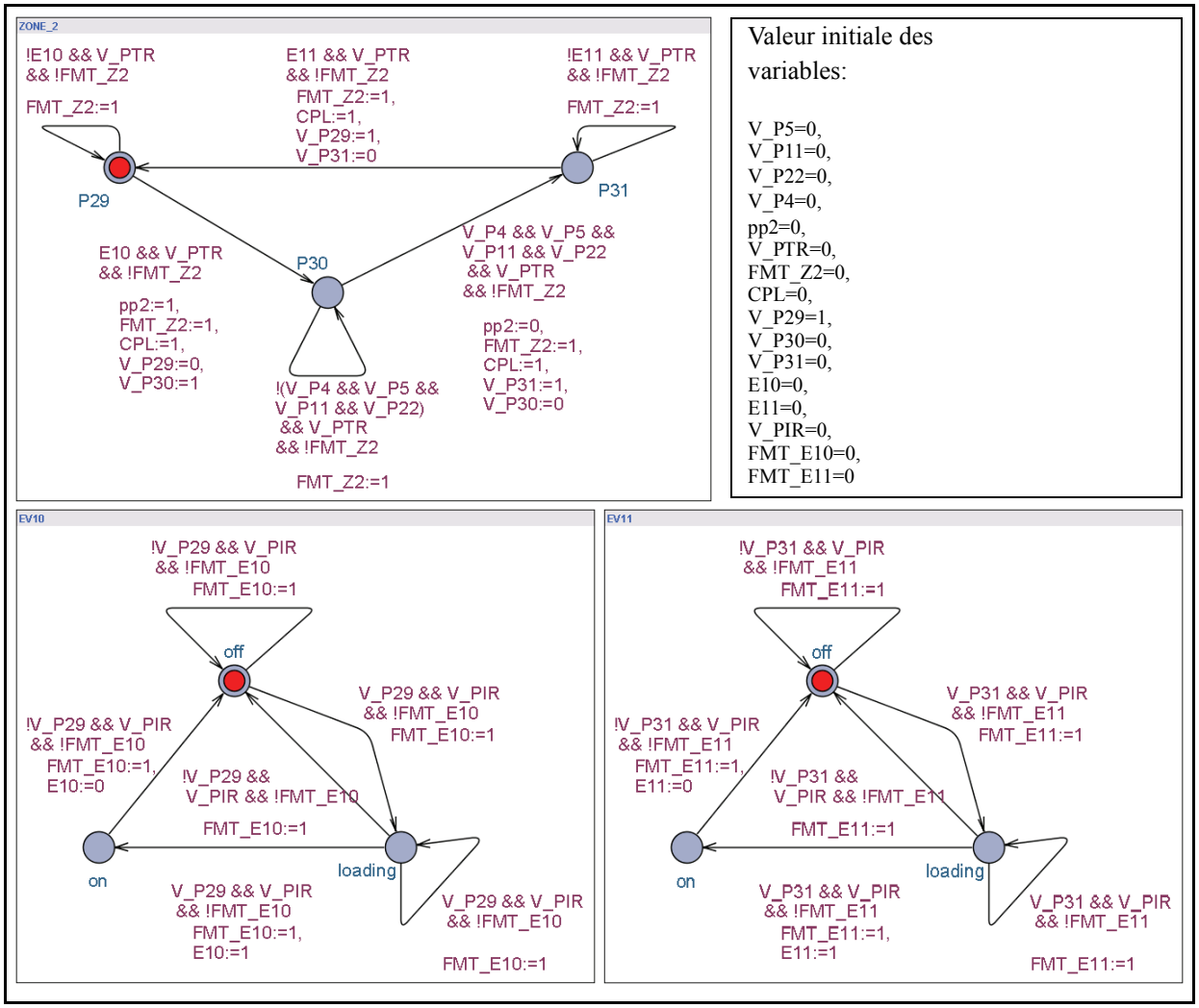


Figure 62: Modèle complet de la ZONE_2.

Les modèles analogues pour les capteurs pp0, pp1 et pp3 sont présentés dans l'annexe 1.

4.5.2.5 Module ZONE_V

Ce module ZONE_V est composé par une chaîne fonctionnelle constituée d'un distributeur monostable (actionné par l'ordre «VENTURI»), d'un capteur de vide («vacuum») et d'une ventouse.

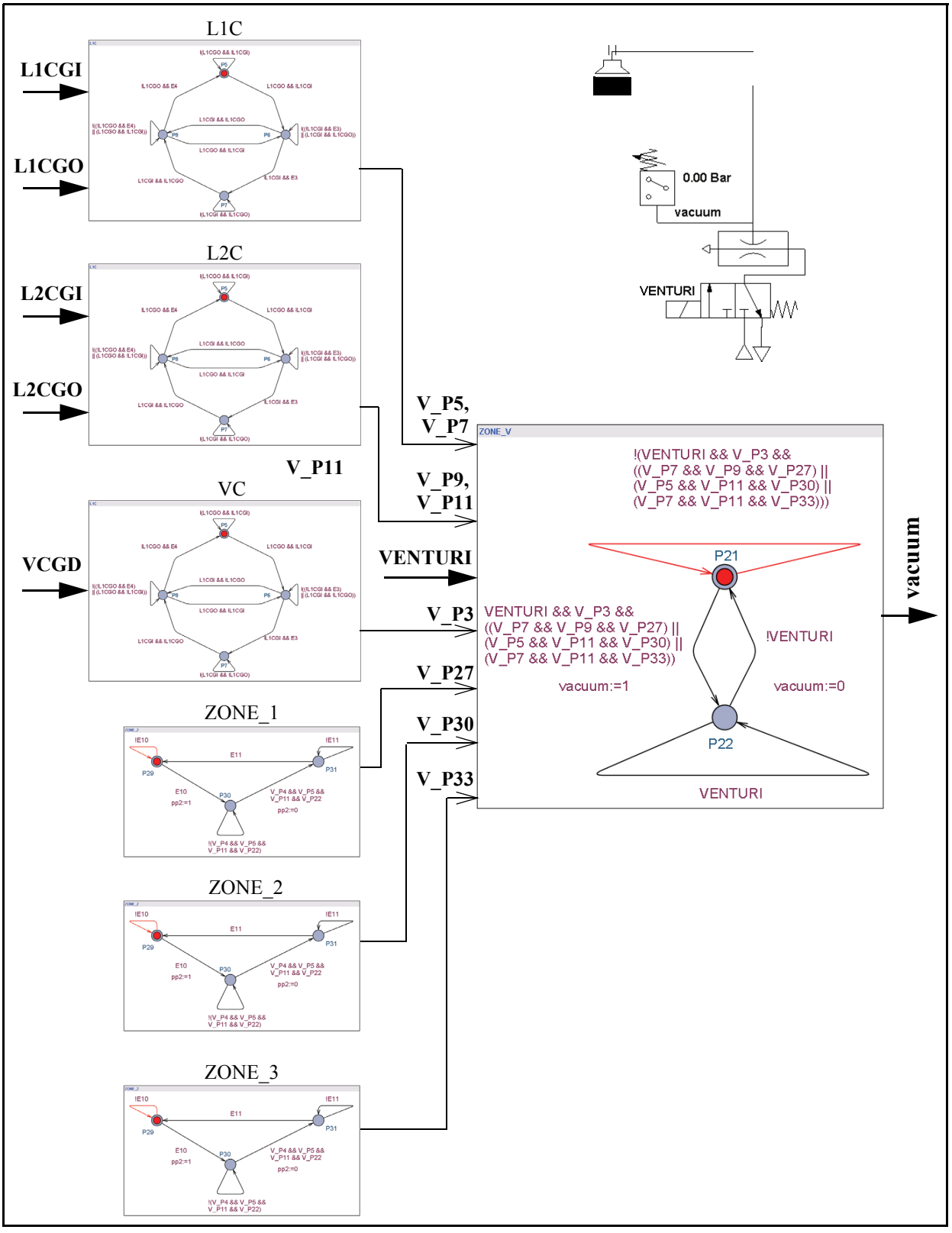


Figure 63: Modèle simplifié de la ZONE_V.

Le comportement de ce module dépend du comportement d'autres modules, comme par exemple celui des ZONE_i (présence ou absence de pièce), des modules VC, L1C et L2C pour que le positionnement de la ventouse soit correct pour la prise d'une pièce, ...

Dans la figure 63 on présente un modèle simplifié du module ZONE_V et sa dépendance d'autres modules. Dans ce cas, de la même manière que pour le module ZONE_2, ce n'est pas seulement le capteur ou le produit que nous devons modéliser individuellement, mais l'ensemble des deux.

Du point de vue du processus, les entrées du module ZONE_V sont des variables générées par d'autres modules du processus (V_P5, V_P7, V_P9, V_P11, V_P3, V_P27, V_P30, V_P33) et une variable d'entrée du processus (VENTURI). Sa sortie est «vacuum».

- P21 : Place qui modélise l'absence de pièce aspirée;
- P22 : Place qui modélise l'aspiration d'une pièce;

Pour l'obtention du modèle complet on a dû ajouter :

- V_PTR, CPL et FMT_ZV pour la synchronisation entre le module ZONE_V et le séquenceur du processus; figure 64.

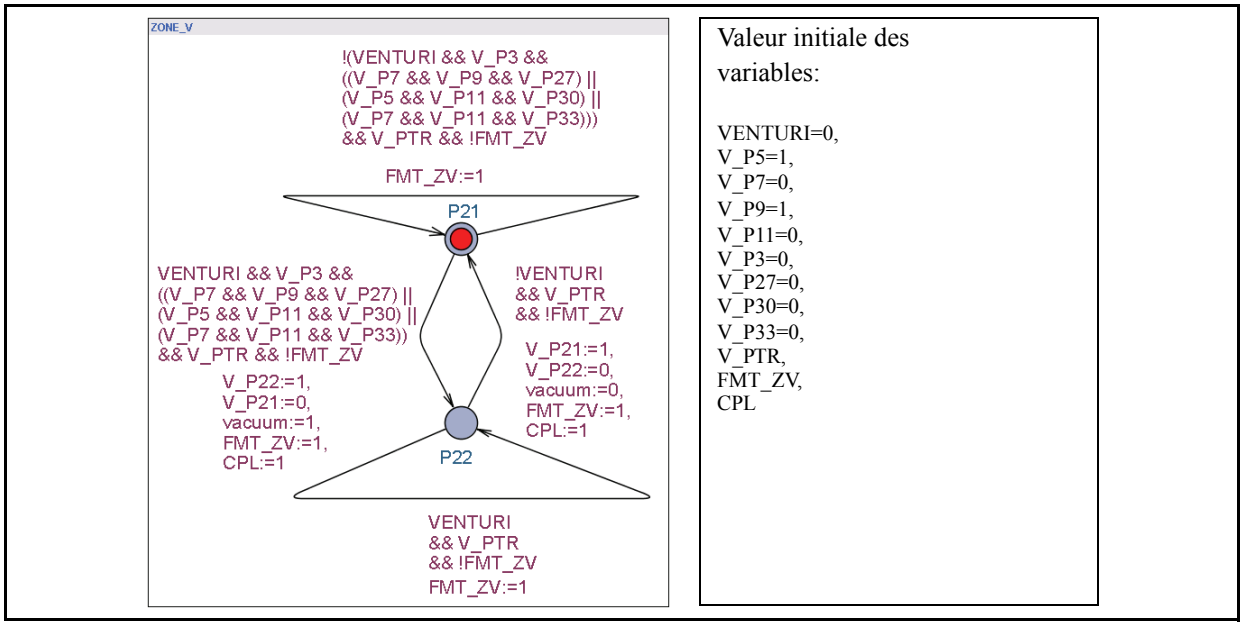


Figure 64: Modèle complet de la «Zone_V».

4.5.2.6 Bilan sur la structure du modèle du processus

On peut constater que la modélisation du processus correspondant à l'exemple support est caractérisée par une difficulté inhérente à l'inter-dépendance des modules du processus. Une représentation graphique de ces dépendance est donnée sur la (figure 65).

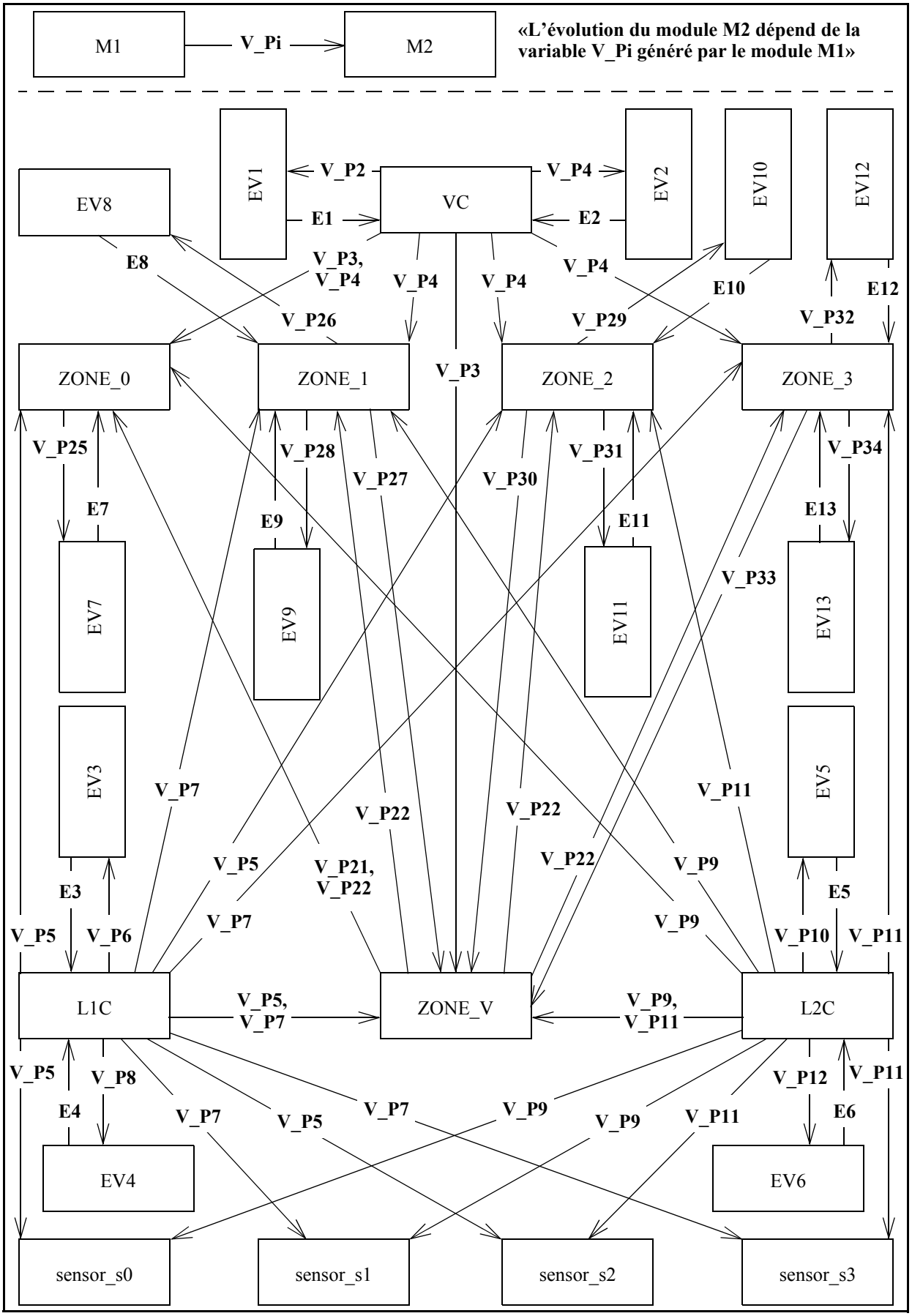


Figure 65: Inter-dépendance entre les modules du processus de l'exemple support.

Comme illustré dans la figure précédente, la modélisation d'un processus peut être rendue complexe par la quantité de variables qui servent à la communication entre modules pour l'évolution du modèle.

4.5.2.7 Séquenceur du processus

Le séquenceur du processus doit assurer la coordination des évolutions des modules du processus (VC, L1C, L2C, sensor_s0, sensor_s1, sensor_s2, sensor_s3, ZONE_0, ZONE_1, ZONE_2, ZONE_3 et ZONE_V) et des évolutions des modules de temporisations logiques (EV1, EV2, EV3, EV4, EV5, EV6, EV7, EV8, EV9, EV10, EV11, EV12, EV13).

Pour la synchronisation avec VC, L1C, L2C, sensor_s0, sensor_s1, sensor_s2, sensor_s3, ZONE_0, ZONE_1, ZONE_2, ZONE_3 et ZONE_V, on utilise respectivement les variables FMT_VC, FMT_L1C, FMT_L2C, FMT_s0p, FMT_s1p, FMT_s2p, FMT_s3p, FMT_Z0, FMT_Z1, FMT_Z2, FMT_Z3 et FMT_ZV. La variable CPL est utilisée par tous les modules.

Pour la synchronisation avec EV1, EV2, EV3, EV4, EV5, EV6, EV7, EV8, EV9, EV10, EV11, EV12 et EV13, on utilise respectivement les variables FMT_E1, FMT_E2, FMT_E3, FMT_E4, FMT_E5, FMT_E6, FMT_E7, FMT_E8, FMT_E9, FMT_E10, FMT_E11, FMT_E12 et FMT_E13.

Le modèle complet du séquenceur de processus, obtenu par instanciation du modèle générique dont la construction à été détaillée dans le chapitre 3, est présenté figure 66.

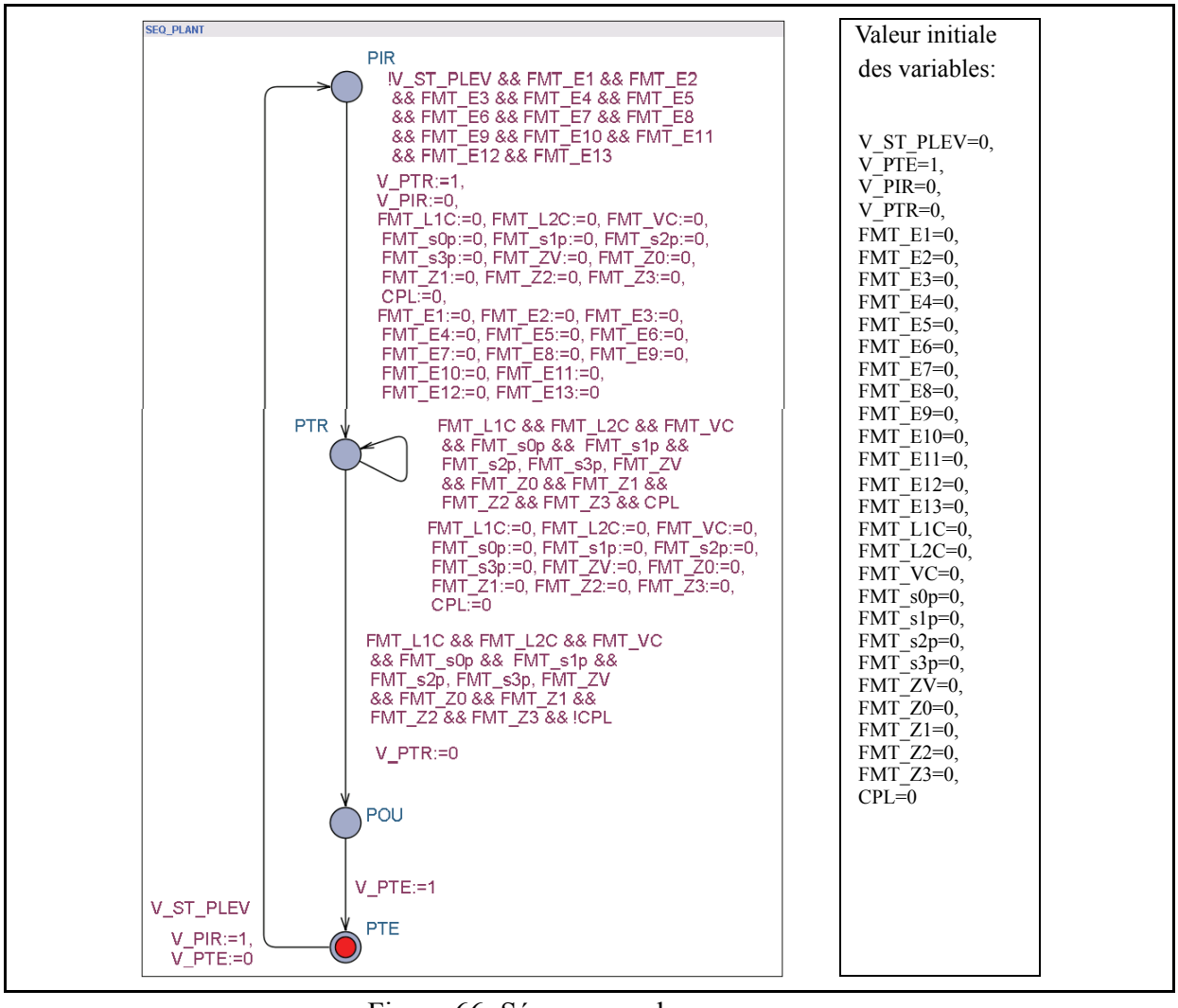


Figure 66: Séquenceur du processus.

4.5.3 Modèle des propriétés à prouver

La liste de propriétés comportementales énoncées dans le paragraphe 4.4 comporte des propriétés dont la formalisation en logique temporelle ne pose pas de problème majeur. Cependant, des propriétés telles que PR_4 («Après avoir été saisie, une pièce ne peut être lâchée qu'au droit de sa position de dépose») nécessitent un examen plus détaillé. Nous allons montrer un peu plus loin qu'une manière efficace pour traduire cette propriété est d'utiliser un automate observateur (qui ajoutera un module à notre modèle global).

4.5.3.1 Propriétés formalisées en logique temporelle

Les propriétés que nous avons formalisées directement en logique temporelle sont présentées dans le tableau 2. Certaines de ces propriétés ont été scindées en plusieurs propriétés plus élémentaires de manière à favoriser à la fois la traduction en logique temporelle et l'obtention d'une preuve plus précise, c'est le cas, par exemple, des propriétés PR_6 et PR_7.

Tableau 2: Formalisation des propriétés de l'exemple support, en logique temporelle

	Propriété en langage naturel	Propriété formalisée en logique temporelle
PR_1 : (PR_1.1, PR_1.2)	«Les ordres envoyés au distributeur qui commande le vérin L1C ne doivent jamais être émis en même temps»; «Les ordres envoyés au distributeur qui commande le vérin L2C ne doivent jamais être émis en même temps»	$AG\neg(L1CGO \wedge L1CGI)$ $AG\neg(L2CGO \wedge L2CGI)$
PR_2	«Si l'ordre de descendre le vérin vertical est présent, alors on ne doit pas avoir l'émission d'ordres de commande relatifs aux mouvements des vérins horizontaux»	$AG(VCGD \Rightarrow \neg(L1CGI \vee L1CGO \vee L2CGI \vee L2CGO))$
PR_3	«S'il y a émission d'un ordre de commande correspondant aux mouvements des vérins horizontaux alors le capteur «vcu» doit être à «1»»	$AG(L1CGI \vee L1CGO \vee L2CGI \vee L2CGO \Rightarrow vcu)$
PR_5	«Il n'y a de mouvements horizontaux que si le capteur «vcu» est à «1»»	$AG(V\_P6 \vee V\_P8 \vee V\_P10 \vee V\_P12 \Rightarrow vcu)$
PR_6 : (PR_6.1 à PR_6.4, PR_6.10, PR_6.11, PR_6.20, PR_6.21, PR_6.30, à PR_6.38)	«Le modèle du contrôleur est toujours vivant»	$AG(X1 \Rightarrow EF\neg X1)$ ... $AG(X4 \Rightarrow EF\neg X4)$ $AG(X10 \Rightarrow EF\neg X10)$ $AG(X11 \Rightarrow EF\neg X11)$ $AG(X20 \Rightarrow EF\neg X20)$ $AG(X21 \Rightarrow EF\neg X21)$ $AG(X30 \Rightarrow EF\neg X30)$ ... $AG(X38 \Rightarrow EF\neg X38)$
PR_7 :	«Si une pièce est détectée par pp1, pp2 ou pp3, alors on aura forcement, dans le futur, la sortie d'un des vérins horizontaux»	
PR_7.1	«Si une pièce est détectée par pp1, alors on aura forcement, dans le futur, la sortie du vérin LC1»	$AG(pp1 \Rightarrow EF(V\_P6))$
PR_7.2	«Si une pièce est détectée par pp2, alors on aura forcement, dans le futur, la sortie du vérin LC2»	$AG(pp2 \Rightarrow EF(V\_P10))$
PR_7.3	«Si une pièce est détectée par pp3, alors on aura forcement, dans le futur, la sortie des vérins LC1 et LC2»	$AG(pp3 \Rightarrow EF(V\_P6 \wedge V\_P10))$

Tableau 2: Formalisation des propriétés de l'exemple support, en logique temporelle

	Propriété en langage naturel	Propriété formalisée en logique temporelle
PR_8 :	«Si une pièce est détectée par pp1, pp2 ou pp3, alors on aura forcement, dans le futur, l'aspiration de la pièce»	
PR_8.1	«Si une pièce est détectée par pp1, alors on aura forcement, dans le futur, l'aspiration de la pièce correspondante»	$AG(pp1 \Rightarrow EF(s1 \wedge vcd \wedge vacuum))$
PR_8.2	«Si une pièce est détectée par pp2, alors on aura forcement, dans le futur, l'aspiration de la pièce correspondante»	$AG(pp2 \Rightarrow EF(s2 \wedge vcd \wedge vacuum))$
PR_8.3	«Si une pièce est détectée par pp3, alors on aura forcement, dans le futur, l'aspiration de la pièce correspondante»	$AG(pp3 \Rightarrow EF(s3 \wedge vcd \wedge vacuum))$
PR_9	«Quand le vérin vertical descend, tous les autres vérins sont et restent en fin de course»	$AG(V\_P2 \Rightarrow (V\_P5 \wedge V\_P9) \vee (V\_P5 \wedge V\_P11) \vee (V\_P7 \wedge V\_P9) \vee (V\_P7 \wedge V\_P11))$

L'expression formelle des propriétés à prouver doit d'une part être la traduction fidèle des propriétés exprimées en langage naturel et d'autre part intégrer les états du modèle où la preuve des propriétés doit être faite :

- après l'évolution du modèle du contrôleur (place PEAC du séquenceur général active, place CTE du séquenceur du contrôleur active, place PTE du séquenceur du processus active et place PTR du séquenceur des propriétés active), et
- après l'évolution du modèle du processus (place PEAP du séquenceur général active, place CTE du séquenceur du contrôleur active, place PTE du séquenceur du processus active et place PTR du séquenceur des propriétés active).

Ces états du modèle sont caractérisés par diverses variables d'état des séquenceurs qui nous permettent de définir les deux variables suivantes qui correspondent respectivement aux deux états stables requis pour les preuves :

$$PEAC = V\_CTE \wedge V\_CPE \wedge V\_TPR \wedge V\_PEAC$$

$$PEAP = V\_CTE \wedge V\_CPE \wedge V\_TPR \wedge V\_PEAP$$

La formalisation complète des propriétés est donnée tableau 3.

Tableau 3: Formalisation des propriétés aux états stables du modèles (PEAC et PEAP)

	Propriétés exprimées en logique temporelle aux états stable PEAC et PEAP
PR_1.1	$AG\neg(L1CGO \wedge L1CGI \wedge (PEAC \vee PEAP))$
PR_1.2	$AG\neg(L2CGO \wedge L2CGI \wedge (PEAC \vee PEAP))$
PR_2	$AG((VCGD \wedge (PEAC \vee PEAP)) \Rightarrow \neg(L1CGI \vee L1CGO \vee L2CGI \vee L2CGO))$
PR_3	$AG(((L1CGI \vee L1CGO \vee L2CGI \vee L2CGO) \wedge (PEAC \vee PEAP)) \Rightarrow vcu)$
PR_5	$AG(((V\_P6 \vee V\_P6 \vee V\_P6 \vee V\_P6) \wedge (PEAC \vee PEAP)) \Rightarrow vcu)$
PR_6	$AG(X1 \Rightarrow EF\neg X1)$, ..., $AG(X4 \Rightarrow EF\neg X4)$, $AG(X10 \Rightarrow EF\neg X10)$, $AG(X11 \Rightarrow EF\neg X11)$, $AG(X20 \Rightarrow EF\neg X20)$, $AG(X21 \Rightarrow EF\neg X21)$, $AG(X30 \Rightarrow EF\neg X30)$, ..., $AG(X38 \Rightarrow EF\neg X38)$

Tableau 3: Formalisation des propriétés aux états stables du modèles (PEAC et PEAP)

	Propriétés exprimées en logique temporelle aux états stable PEAC et PEAP
PR_7.1	$AG((pp1 \wedge (PEAC \vee PEAP)) \Rightarrow EF(V\_P6 \wedge (PEAC \vee PEAP)))$
PR_7.2	$AG((pp2 \wedge (PEAC \vee PEAP)) \Rightarrow EF(V\_P10 \wedge (PEAC \vee PEAP)))$
PR_7.3	$AG((pp3 \wedge (PEAC \vee PEAP)) \Rightarrow EF(V\_P6 \wedge V\_P10 \wedge (PEAC \vee PEAP)))$
PR_8.1	$AG((pp1 \wedge (PEAC \vee PEAP)) \Rightarrow EF(s1 \wedge vcd \wedge vacuum \wedge pp1 \wedge (PEAC \vee PEAP)))$
PR_8.2	$AG((pp2 \wedge (PEAC \vee PEAP)) \Rightarrow EF(s2 \wedge vcd \wedge vacuum \wedge pp1 \wedge (PEAC \vee PEAP)))$
PR_8.3	$AG((pp3 \wedge (PEAC \vee PEAP)) \Rightarrow EF(s3 \wedge vcd \wedge vacuum \wedge pp1 \wedge (PEAC \vee PEAP)))$
PR_9	$AG((V\_P2 \wedge (PEAC \vee PEAP)) \Rightarrow$ $((V\_P5 \wedge V\_P9) \vee (V\_P5 \wedge V\_P11) \vee (V\_P7 \wedge V\_P9) \vee (V\_P7 \wedge V\_P11)))$

4.5.3.2 Formalisation des propriétés séquentielles

La seule propriété dont la formalisation en logique temporelle ne nous semble pas aisée est la propriété (PR_4 : «Après avoir été saisie, une pièce ne peut être lâchée qu'au droit de sa position de dépose»). Pour simplifier la compréhension du modèle de propriété que nous allons maintenant élaborer, on adoptera la désignation «PICK» («*pick position*») pour indiquer la position de prise d'une pièce et «PLACE» («*place position*») pour indiquer la position de dépose d'une pièce. Le modèle simplifié de cette propriété est proposé figure 67.

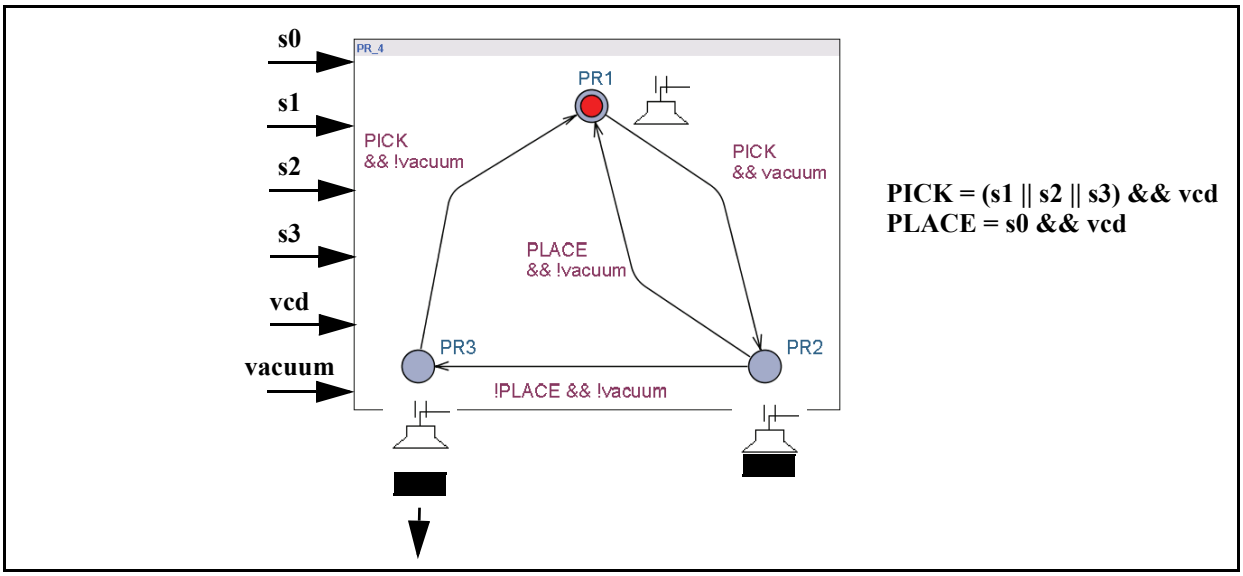


Figure 67: Modèle simplifié du module PR_4, correspondant à la propriété séquentielle PR_4.

Les entrées de ce module PR_4 sont «s0», «s1», «s2», «s3», «vcd» et «vacuum».

- PR1 : Place qui modélise l'absence de pièce aspirée quand la ventouse ne se trouve ni dans l'une des position de prise ni dans la position de dépose;
- PR2 : Place qui modélise la présence d'une pièce aspirée et non encore déposée;
- PR3 : Place qui modélise un comportement non désiré (la pièce a été lâchée entre l'une des positions de prise et la position de dépose);

Pour l'obtention du modèle complet on a dû ajouter:

- V_DPR, CPRL et FPRT_PR_4 pour la synchronisation entre le module PR_4 et le séquenceur des propriétés; figure 68.

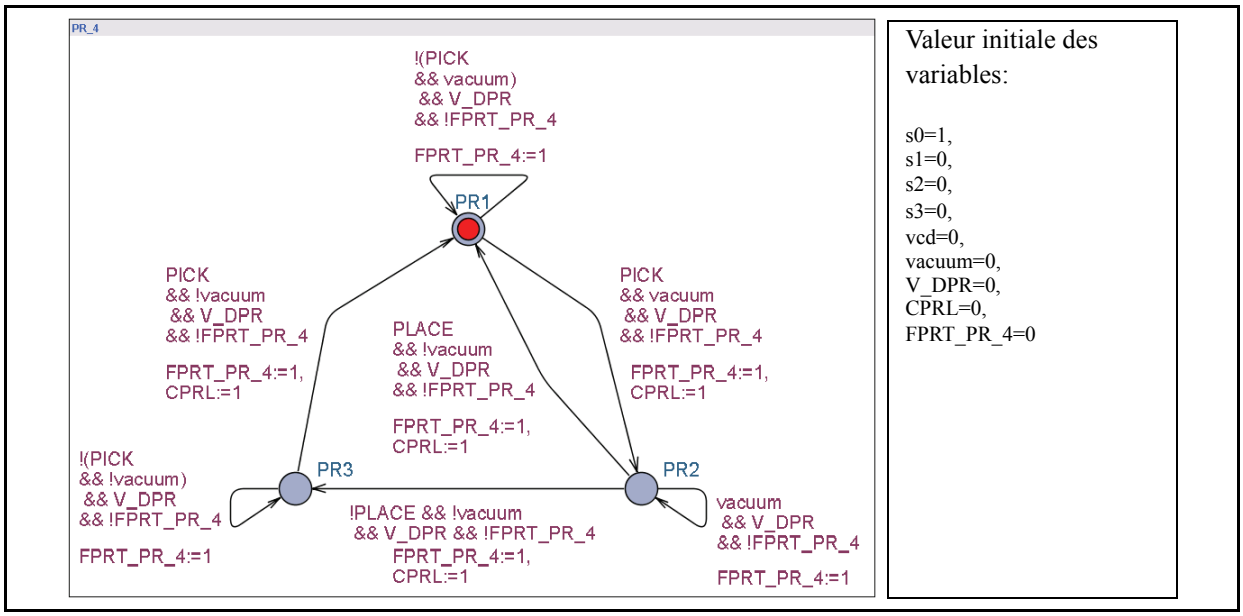


Figure 68: Modèle complet du module PR_4, correspondant à la propriété séquentielle PR_4.

Ainsi doté d'un automate traduisant facilement la sémantique séquentielle de la propriété PR_4 et d'un état observateur du non respect de cette propriété, on peut facilement exprimer en logique temporelle que la situation non désirée n'arrive jamais aux deux situations stables PEAC et PEAP par :

- PR_4 : $AG\neg(PR3 \wedge (PEAC \vee PEAP))$. «La place PR3, du modèle des propriétés, ne doit jamais être active pendant les deux situations de preuve».

4.5.3.3 Séquenceur des propriétés

Le séquenceur des propriétés doit assurer la coordination des évolutions du module de la propriété PR_4. Pour la synchronisation avec PR_4 il utilise les variables FPRT_PR4 et CPRL. Le modèle complet, obtenu par instanciation du modèle générique dont la construction a été décrite en 4.2, est présenté figure 69.

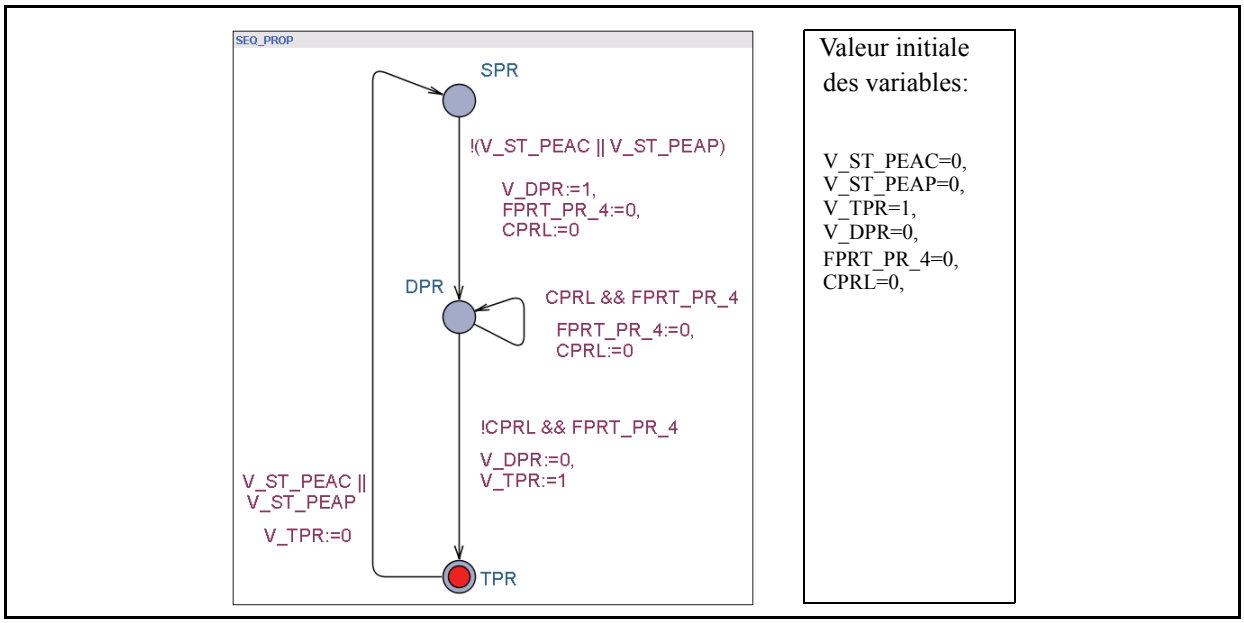


Figure 69: Séquenceur des propriétés de l'exemple support.

4.5.4 Séquenceur général

Le séquenceur général étant complètement générique, il ne nécessite pas de particularisation pour une étude de cas particulière. Il est donc utilisé tel que défini en "Séquenceur général", page 65 et illustré figure 50.

4.6 Bilan du chapitre 4

Nous avons montré dans ce chapitre comment obtenir le modèle de l'exécution d'un contrôleur, spécifié en SFC et exécuté par un équipement de contrôle cyclique. Nous avons pour cela défini un séquenceur du contrôleur. Puis, une technique de traduction de propriétés complexes, baptisées «séquentielles», en utilisant conjointement une logique temporelle et un automate à états, à été définie. Nous avons également défini et construit un séquenceur des propriétés. Enfin, un séquenceur général a pu être introduit de manière à ce que les interactions entre processus et contrôleur soient correctement traduites et que la vérification des propriétés sur état stable du système automatisé puissent être garantie.

La mise en oeuvre de l'ensemble de ces résultats sur un exemple dont nous avons réalisé une modélisation complète a permis de mettre en évidence la généricité de notre approche et la réutilisabilité de nos modèles.

Dans le chapitre suivant seront présentés les résultats de la vérification formelle du modèle que nous venons de construire. Nous pourrons ainsi conclure quant à la pertinence de notre approche pour la vérification par model-checking avec prise en compte d'un modèle de processus.

Chapitre 5
Étude comparée de la vérification formelle avec ou sans modèle de processus

À partir d'un modèle prêt à être vérifié, un algorithme de traduction en langage d'entrée du model-checker NuSMV est proposé en début de chapitre. Les résultats expérimentaux de vérification formelle de notre étude de cas sont présentés et commentés selon plusieurs points de vue pour en faire émerger des règles plus générales sur les deux approches «Non Model-Based» et «Model-Based» en fonction de la nature des propriétés à vérifier. Il sera alors possible de proposer une stratégie de vérification des propriétés tirant le meilleur parti des deux approches afin de maximiser la pertinence des résultats de preuve.

Avant de présenter les résultats expérimentaux de la vérification formelle puis de comparer les deux approches Non Model-Based (NMB) et Model-Based (MB), ce chapitre présente une technique systématique, comportant cinq règles, pour traduire le modèle complet du système à vérifier en code d'entrée du model-checker retenu : NuSMV. Puis, les résultats obtenus lors de la vérification formelle seront présentés.

5.1 Traduction du modèle en code d'entrée du model-checker NuSMV

NuSMV, le Model-Checker retenu, permet la vérification d'un système d'automates finis vis-à-vis de propriétés exprimées en logique temporelle. Son langage d'entrée assure la description des systèmes d'automates allants de configurations entièrement synchrones jusqu'à des configurations entièrement asynchrones. Le modèle sous-jacent est une structure finie de Kripke qui est un automate à états finis non déterministe dont les états ont des labels qui sont des expressions à valeur booléenne.

Afin d'exprimer notre modèle sous la forme d'une structure de Kripke avec la syntaxe de NuSMV nous proposons un algorithme basé sur cinq règles de traduction énoncées ci-après.

5.1.1 Règle 1 : Structure générale du code d'entrée de NuSMV

NuSMV intègre la notion de description modulaire du modèle à vérifier. Cependant, l'utilisation de cette primitive appelée «module», introduit dans le modèle des variables supplémentaires dédiées à la définition des entrées et des sorties de chaque module de NuSMV. Une telle augmentation du nombre de variables a un impact direct sur l'explosion combinatoire lors de la phase de vérification. Pour limiter l'explosion combinatoire lors de la vérification de notre modèle une structure «à plat» du code à donc été retenue. Le code NuSMV se réduit au seul module «main»

```
MODULE main
      ........

      ........
```

Le contenu du module est alors une succession de sections de type «VAR» et «IVAR» pour la déclaration des variables, «ASSIGN» pour l'assignation des variables, «DEFINE» pour la définition d'alias et «SPEC» pour l'expression des propriétés à vérifier.

5.1.2 Règle 2 : Traduction de la structure d'un module du modèle

Cette règle définit comment la structure place-transition (incluant les gardes) d'un module est traduite de façon systématique en NuSMV. Considérons l'exemple de la figure 70, pour illustrer cette règle.

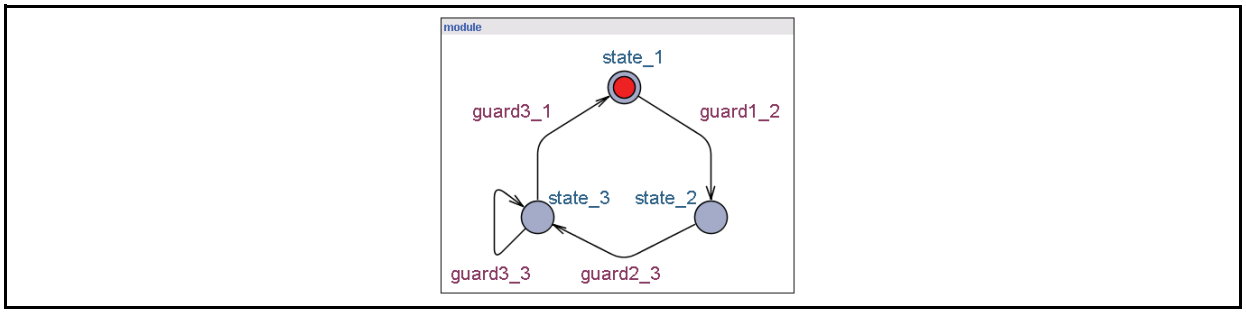


Figure 70: Exemple d'un module (nommé «module»).

L'état actif est codé par une variable de type énumérée («scalar variable») qui peut prendre différentes valeurs symboliques :

```
VAR
     module {state_1, state_2, state_3};
```

L'état initial de cette variable est codé par une assignation initiale :

```
ASSIGN
     init(module) := state_1;
```

Toutes les transitions d'un module sont codées par une seule structure «*case ... esac*» où le nombre de cas à traiter est égal au nombre de transitions plus une (le cas par défaut) :

```
ASSIGN
     next(module) := case
               module=state_1 & guard1_2 : state_2;
               module=state_2 & guard2_3 : state_3;
               module=state_3 & guard3_3 : state_3;
               module=state_3 & guard3_1 : state_1;
               1 : module;
          esac;
```

Les variables associées aux places de notre modèle n'ont pas fait l'objet d'une déclaration dans le code NuSMV puisque les variables «module» assurent déjà cette fonction. La valeur de l'activité de la place «*state_2*» est le résultat du test «*module=state_2*». Pour conserver la notation «*V_state_2*» utilisée dans le modèle, nous ajoutons dans le code NuSMV, la définition d'un alias comme suit pour chaque état du module :

```
DEFINE
     V_state_2 := (module=state_2);
```

Grâce à cet alias nous pourrons conserver notre notation sans pour autant augmenter le nombre de variables définies dans le code NuSMV. Cela contribue au contrôle de l'explosion combinatoire de la vérification des propriétés.

5.1.3 Règle 3 : Traduction des assignations des variables du modèle

Chaque variable du modèle (hormis les variables associées à l'activité d'une place, qui sont prises en compte par la règle 2) est traitée de façon identique dans le code NuSMV : une déclaration, une assignation initiale et une assignation de type «*next*». On rappelle que le formalisme retenu ne contient que des variables logiques. Soit $\{tr_i\}_{i \in [1,n]}$ l'ensemble des transitions du modèle qui assignent une valeur à la variable V ; Soit P_i la place source de la transition tr_i dont l'activité est définie par la variable V_P_i ; Soit G_i la garde de la transition tr_i ; Soit exp_i l'expression assignée à la variable V, au tir de tr_i et soit V^0 la valeur initiale de V. La traduction de la variable est la suivante :

```
VAR
     V : boolean;
ASSIGN
     init(V) := V^0
     next(V) := case
               V_P_1 & G_1 : exp_1;
               V_P_2 & G_2 : exp_2;
               ...
               V_P_n & G_n : exp_n;
               1 : V;
          esac;
```

5.1.4 Règle 4 : Gestion de la simultanéité de franchissement des transitions

La classe d'automates que nous avons définie exclut le franchissement simultané de plusieurs transitions. Or avec NuSMV rien ne s'oppose implicitement à des franchissements simultanés. Il est donc nécessaire d'ajouter un mécanisme explicite de non franchissement simultané dans la traduction en code NuSMV.

Pour ce faire, l'idée de base consiste à numéroter de manière unique chaque transition de chaque module du modèle, puis à déclarer une variable entière (notée FR) comprise entre 1 et le «nombre de transitions dans tout le modèle» (noté NT_{max}) dans le code NuSMV. Cette variable FR restera une variable libre, c'est-à-dire sans assignation dans le code NuSMV. Elle pourra donc prendre n'importe quelle valeur au cours de la vérification, mais bien entendu une seule valeur à la fois. Comme nous l'avons vu dans la règle de traduction d'un module de l'automate : les transitions sont codées par des cas dans une structure «case ... esac». Il suffit d'enrichir chaque cas pour que la transition ne soit franchissable que lorsque, en plus de la condition de franchissement de base, FR est égale au numéro de la transition. A titre d'exemple, la figure 71 propose une numérotation arbitraire des transitions du module de la figure 70.

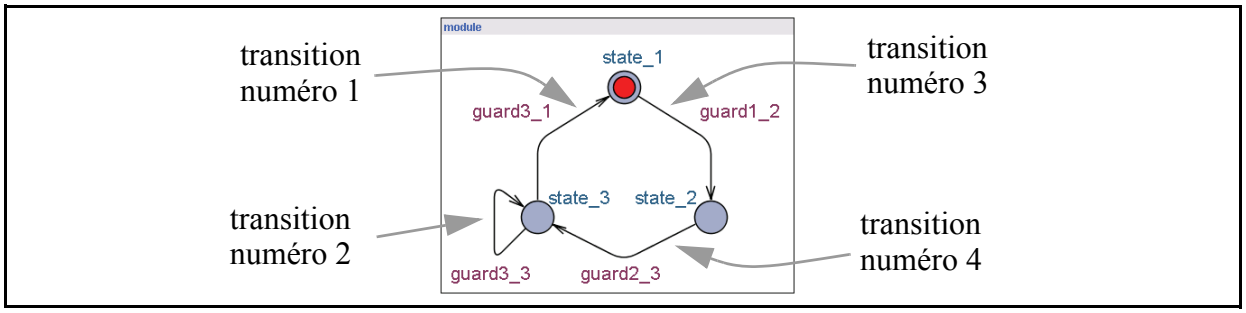


Figure 71: Exemple d'un module avec une numérotation arbitraire des transitions.

Le code NuSMV inclut donc une nouvelle déclaration pour la variable FR :

```
IVAR
    FR : 1..NT_max;
```

et les structures «case ... esac» de chaque module sont enrichies comme suit :

```
ASSIGN
    next(module) := case
                module=state_1 & guard1_2 & (FR = 3) : state_2;
                module=state_2 & guard2_3 & (FR = 4) : state_3;
                module=state_3 & guard3_3 & (FR = 2) : state_3;
                module=state_3 & guard3_1 & (FR = 1) : state_1;
                1 : module;
            esac;
```

Cette procédure systématique a pour effet de faire croitre l'espace d'états potentiel du modèle à vérifier puisque la variable FR peut prendre n'importe quelle valeur dans l'intervalle $[1, NT_{max}]$. Pour en limiter les effets pénalisants durant la vérification, nous avons mis en place une utilisation plus optimale de la variable FR en n'affectant un numéro distinct qu'aux seules transitions qui peuvent évoluer simulatnément.

5.1.5 Règle 5 : Traduction des propriétés en code NuSMV

Pour les propriétés exprimées avec un automate observateur, la traduction s'effectue de manière similaire à tous les autres modules en appliquant les règles de traduction précédemment citées. Con-

cernant les aspects exprimés en logique temporelle, la traduction se limite à une simple adaptation de la syntaxe. Par exemple :

$AG\ \neg(L1CGO \wedge L1CGI)$ devient

```
SPEC
     AG !(L1CGI & L1CGO);
```

$AG\ (VCGD \Rightarrow \neg(L1CGI \vee L1CGO \vee L2CGI \vee L2CGO))$ devient

```
SPEC
     AG (VCGD -> !(L1CGI | L1CGO | L2CGI | L2CGO));
```

$AG(pp3 \Rightarrow EF(V\text{–}P6 \wedge V\text{–}P10))$ devient

```
SPEC
     AG (pp3 -> EF (V_P6 & V_P10)).
```

5.1.6 Algorithme de génération du code NuSMV

A partir des règles qui viennent d'être exprimées, la génération du code NuSMV traduisant le comportement du modèle et des propriétés à vérifier est obtenu en appliquant l'algorithme suivant :

```
-- Préparation de la structure générale du code
Appliquer la règle 1

-- Traduction du comportement des modules
Pour tous les modules du modèle (incluant les automates observateurs)
     Appliquer la règle 2

-- Affectation des variables du modèle
Pour toutes les variables du modèle hormis les variables d'état des places (V_x)
     Appliquer la règle 3

-- Traduction des propriétés
Pour toutes les propriétés
     Appliquer la règle 5

-- Adaptation du code pour éviter les franchissements simultanés
Appliquer la règle 4 au code NuSMV précédemment généré.
```

Pour notre étude de cas développée dans le chapitre 4, le code NuSMV ainsi généré est présenté in extenso dans l'annexe 2.

5.2 Vérification des propriétés avec les deux approches NMB et MB.

Les vérifications formelles ont été conduites avec une machine disposant d'un processeur Pentium III cadencé à 1 GHz, d'une capacité mémoire vive de 1 Go, et avec la version 2.1.2 du model-checker NuSMV.

Le model-checker indique que :

- Le modèle NMB atteint 6,1 x 10^6 états sur les 2,6 x 10^{24} états possibles;
- Le modèle MB atteint 5,0 x 10^8 états sur les 3,9 x 10^{47} états possibles;

Le tableau 4 présente les résultats obtenus pour les deux approches NMB et MB (véracité des propriétés et temps de calcul pour obtenir la preuve). Pour chaque propriété, il est rappelé son libellé en langage naturel ainsi que son type (S-sûreté, V-vivacité). Les propriétés qui ne peuvent pas être exprimées par l'approche NMB (car elles sont exprimées à l'aide de variables de processus) sont repérées dans le tableau par la notation «X».

Tableau 4: Résultats obtenus pour les deux approches NMB et MB.

Propriété				Résultats expérimentaux			
Nom	Enoncé de la propriété	Type	Variables	Non Model-Based (NMB)		Model-Based (MB)	
				Preuve	Temps de calcul	Preuve	Temps de calcul
PR_1	Les ordres envoyés au distributeur qui commande le vérin L1C ne doivent jamais être émis en même temps	S	C	vraie	7 s	vraie	134 min
PR_2	Si l'ordre de descendre le vérin vertical est présent, alors on ne doit pas avoir l'émission d'ordres de commande relatifs aux mouvements des vérins horizontaux	S	C	fausse	13 s	vraie	136 min
PR_3	S'il y a émission d'un ordre de commande correspondant aux mouvements des vérins horizontaux alors le capteur «vcu» doit être à «1»	S	C, O	fausse	9 s	vraie	135 min
PR_4	Après avoir été saisie, une pièce ne peut être lâchée qu'au droit de sa position de dépose	S	O	fausse	9 s	vraie	136 min
PR_5	Il n'y a de mouvements horizontaux que si le capteur «vcu» est à «1»	S	O, P	X	X	vraie	135 min
PR_6	Le modèle du contrôleur est toujours vivant	V	C	vraie	15 s	fausse	640 min
PR_7.1	Si une pièce est détectée par pp1, pp2 ou pp3, alors on aura forcement, dans le futur, la sortie d'un des vérins horizontaux	V	O, P	X	X	fausse	532 min
PR_7.2		V	O, P	X	X	vraie	237 min
PR_7.3		V	O, P	X	X	fausse	510 min
PR_8.1	Si une pièce est détectée par pp1, pp2 ou pp3, alors on aura forcement, dans le futur, l'aspiration de la pièce	V	O	vraie	8 s	fausse	486 min
PR_8.2		V	O	vraie	8 s	fausse	453 min
PR_8.3		V	O	vraie	8 s	fausse	534 min
PR_9	Quand le vérin vertical descend, tous les autres vérins sont et restent en fin de course	S	P	X	X	vraie	137 min

5.3 Interprétation des résultats obtenus

5.3.1 Remarques générales

En analysant la taille des modèles explorés par le model-checker, on constate dans un premier temps que le nombre des combinaisons possibles pour les valeurs des variables du modèle de l'approche MB est très supérieur à celui de l'approche NMB (3,9 x 10^{47} combinaisons pour MB contre «seulement» 2,6 x 10^{24} pour NMB). Cette énorme différence s'explique tout simplement par le fait que la frontière d'isolement du modèle de l'approche MB est plus large et inclut celle de l'approche NMB. Cela a conduit à l'ajout de 25 modules nécessaires à la modélisation du processus (figure 65), et donc à l'augmentation corrolaire du nombre de combinaisons possibles pour les valeurs des variables du modèle.

La grande interrogation, avant l'obtention de ces résultats expérimentaux, portait sur l'aptitude du couple (ordinateur / model-checker) à mener à leurs termes les preuves de propriétés. Les deux principales sources de problèmes étant :

- la taille de l'espace d'états du modèle par rapport à la mémoire vive que l'ordinateur met à la disposition du model-checker;
- la durée d'exploration du modèle pour conclure sur les propriétés.

L'expérimentation a montré que notre approche MB a permis la maîtrise de l'explosion combinatoire pendant la vérification. En effet, toutes les preuves ont été conduites jusqu'à leur terme malgré 23 ordres de grandeurs d'écart (de 10^{24} à 10^{47}) sur le nombre de combinaisons possibles pour les variables du modèle. Cela s'explique au regard de l'espace des états réellement atteignables par les modèles : 6,1 x 10^6 pour l'approche NMB contre 5,0 x 10^8 pour l'approche MB. Seulement deux ordres de grandeur séparent les deux espaces atteignables, ce qui ne multiplie que par 100 le nombre d'états mis en jeu lors de la vérification des propriétés. Si l'augmentation de l'espace d'états atteignables par l'approche MB reste limitée malgré l'ajout des 25 modules qui modélisent le processus, c'est parce qu'en contre-partie, ce modèle de processus restreint l'espace d'états atteignables par le modèle du contrôleur en réduisant les combinaisons possibles de ses entrées aux seules évolutions réalistes.

5.3.2 Remarques détaillées par propriété

5.3.2.1 Propriété PR_1

«Les ordres envoyés au distributeur qui commande le vérin L1C ne doivent jamais être émis en même temps», et «Les ordres envoyés au distributeur qui commande le vérin L2C ne doivent jamais être émis en même temps» : propriété de sûreté qui ne concerne que des variables contrôlables par le modèle du contrôleur.

Le résultat «PR_1 vraie» sans modèle du processus (NMB) est interprété comme suit :

- Quel que soit le comportement du processus, le contrôleur produit des sorties sûres.

Le résultat «PR_1 vraie» avec modèle du processus (MB) est interprété comme suit :

- Couplé à un comportement nominal du processus, le contrôleur produit des sorties sûres.

Pour cette propriété, l'approche NMB qui ne fait aucune hypothèse sur le comportement du processus fournit un résultat de plus grande portée.

5.3.2.2 Propriété PR_2

«Si l'ordre de descendre le vérin vertical est présent, alors on ne doit pas avoir l'émission d'ordres de commande relatifs aux mouvements des vérins horizontaux» : propriété de sûreté qui ne concerne que des variables contrôlables par le modèle du contrôleur.

Le résultat «PR_2 fausse» sans modèle du processus (NMB) est interprété comme suit :

- Le contrôleur ne garantit pas l'émission sécuritive de l'ordre de descente du vérin lorsque ses entrées peuvent évoluer sans contrainte. Comme cette propriété est exprimée uniquement avec des variables contrôlables par le contrôleur, ce résultat de preuve est interprété comme une erreur de spécification du contrôleur. Un réexamen de la spécification doit permettre d'éliminer ce problème.

Le résultat «PR_2 vraie» avec modèle du processus (MB) est interprété comme suit :

- Couplé à un comportement nominal du processus, le contrôleur conduit l'ensemble du système de façon sûre. Pour ce type de propriétés, l'emploi d'un modèle de processus réduit la portée de la preuve au seul fonctionnement nominal du processus.

Pour cette propriété l'approche NMB, qui ne fait aucune hypothèse sur le comportement du processus, fournit un résultat de plus grande portée et nous suggère de corriger la spécification du contrôleur jusqu'à l'obtention du résultat «PR_2 vraie» sans modèle du processus.

5.3.2.3 Propriétés PR_3 et PR_4

«S'il y a émission d'un ordre de commande correspondant aux mouvements des vérins horizontaux alors le capteur «vcu» doit être à «1»», «Après avoir été saisie, une pièce ne peut être lâchée qu'au droit de sa position de dépose» : propriétés de sûreté qui concernent des variables qui ne sont pas toutes contrôlables par le contrôleur (variables d'entrées du contrôleur).

Le résultat «PR_3 et PR_4 fausses» sans modèle du processus (NMB) est interprété comme suit :

- Le contrôleur ne peut pas à lui seul garantir la valeur de ses entrées (variables non contrôlables). Il n'est dont pas illogique d'obtenir ce résultat, et un réexamen du comportement du contrôleur n'y changerait rien. Par conséquent, il est impossible de faire le moindre lien entre cette preuve et le comportement du contrôleur ou du système complet (contrôleur plus processus).

Le résultat «PR_3 et PR_4 vraies» avec modèle du processus (MB) est interprété comme suit :

- Couplé à un comportement nominal du processus, le contrôleur conduit l'ensemble du système de façon sûre. Pour ce type de propriétés, l'emploi d'un modèle de processus réduit la portée de la preuve au seul fonctionnement nominal du processus.

Pour ces propriétés, seule l'approche MB fournit des résultats utilisables, tout en sachant que la portée de ces résultats est limitée au comportement d'un processus tel qu'il est décrit dans le modèle fourni au model-checker.

5.3.2.4 Propriétés PR_5 et PR_9

«Il n'y a de mouvements horizontaux que si le capteur «vcu» est à «1»», et «Quand le vérin vertical descend, tous les autres vérins sont et restent en fin de course» : propriétés de sûreté exprimées à l'aide de variables de processus (états du modèle du processus).

PR_5 et PR_9 ne peuvent pas être exprimées sans faire appel au comportement du processus. Le model-cheking NMB n'a donc pas de sens.

Le résultat «PR_5 et PR_9 vraies» avec modèle du processus (MB) est interprété comme suit :

- Couplé à un comportement nominal du processus, le contrôleur conduit l'ensemble du système de façon sûre. Pour ce type de propriétés, l'emploi d'un modèle de processus réduit la portée de la preuve au seul fonctionnement nominal du processus.

Pour ces propriétés, seule l'approche MB peut être déroulée jusqu'à son terme. Mais la portée des résultats est limitée au comportement d'un processus tel qu'il est décrit dans le modèle fourni au model-checker.

5.3.2.5 Propriété PR_6

«Le modèle du contrôleur est toujours vivant» : propriété de vivacité qui ne concerne que des variables contrôlables par le modèle du contrôleur.

Le résultat «PR_6 vraie» sans modèle du processus (NMB) est interprété comme suit :

- Le contrôleur est vivace lorsque l'on ne contraint pas les évolutions des entrées du modèle par le comportement du processus. Toutes les évolutions des entrées du contrôleur étant permises, on ne peut pas garantir que le comportement du contrôleur reste vivace lorsqu'il fonctionne en boucle fermée avec le processus. Pour ce type de propriétés l'approche NMB fournit un résultat de portée très limitée. En revanche, il faut noter que si le résultat avait été «PR_6 fausse» sans modèle de processus on aurait immédiatement pu conclure à la nécessité d'un réexamen du comportement du contrôleur.

Le résultat «PR_6 fausse» avec modèle du processus (MB) est interprété comme suit :

- Couplé à un comportement nominal du processus, le contrôleur n'est plus vivace. Les contraintes imposées par le processus vont entraîner le contrôleur dans des situtations blocquantes. Cela est interprété comme une erreur de spécification du comportement du contrôleur. Son réexamen doit permettre d'éliminer ce problème.

Pour ce type de propriétés, l'approche MB permet d'augmenter la portée de la preuve en incluant la détection de blocages du système dus à l'interaction du contrôleur et du processus. Ici le blocage identifié par l'approche MB suggère un réexamen pour correction de la spécification du contrôleur jusqu'à l'obtention du résultat «PR_2 vraie» avec le modèle du processus (approche MB).

5.3.2.6 Propriétés PR_7.1 et PR_7.3

«Si une pièce est détectée par pp1, alors on aura forcement, dans le futur, la sortie du vérin LC1» et «Si une pièce est détectée par pp2, alors on aura forcement, dans le futur, la sortie du vérin LC2» : propriétés de vivacité qui concernent des variables observables et des variables processus (donc non contrôlables).

PR_7.1 et PR_7.3 ne peuvent pas être exprimées sans faire appel au comportement du processus. Le model-cheking NMB n'a donc pas de sens.

Le résultat «PR_7.1 et PR_7.3 fausses» avec modèle du processus (MB) est interprété comme suit :

- Le comportement du contrôleur étant couplé à un comportement nominal du processus les propriétés ne sont pas vérifiées. En faisant l'hypothèse que le comportement du modèle de processus est exempt d'erreur, ce résultat de preuve est interprété comme une erreur de spécification du contrôleur qui n'arrive pas à atteindre une situation pour laquelle il émettra les sorties requises. Une phase de re-conception doit permettre d'éliminer ce problème.

Pour ces propriétés, seule l'approche MB peut être déroulée jusqu'à son terme. De plus elle fournit un résultat utilisable. Après avoir identifié l'absence d'un chemin, l'approche MB suggère un réexamen pour correction de la spécification du contrôleur.

5.3.2.7 Propriété PR_7.2

PR_7.2 ne peut pas être exprimée sans faire appel au comportement du processus. Le model-cheking NMB n'a donc pas de sens.

Le résultat «PR_7.2 vraie» avec modèle du processus (MB) est interprété comme suit :

- Le comportement du contrôleur étant couplé à un comportement nominal du processus la propriété est vérifiée. Le résultat se limite aux systèmes dont le processus se comporte selon le modèle fourni au model-checker.

Pour cette propriété, seule l'approche MB peut être déroulée jusqu'à son terme. De plus elle fournit un résultat significatif.

5.3.2.8 Propriété PR_8.1, PR_8.2 et PR_8.3

«Si une pièce est détectée par pp1, alors on aura forcement, dans le futur, l'aspiration de la pièce correspondante», «Si une pièce est détectée par pp2, alors on aura forcement, dans le futur, l'aspiration de la pièce correspondante» et «Si une pièce est détectée par pp3, alors on aura forcement, dans le futur, l'aspiration de la pièce correspondante» : propriétés de vivacité qui concernent des variables observables par le contrôleur.

Le résultat «PR_8.1, PR_8.2 et PR_8.3 vraies» sans modèle du processus (NMB) est interprété comme suit :

- Pour chacune des ces trois propriétés, le contrôleur est capable de produire au moins une séquence de sorties qui permet d'assurer la preuve. Par contre, ces séquences de sorties sont obtenues sans la moindre restriction sur les évolutions d'entrées. L'approche NMB fournit donc un résultat de portée très limitée.

Le résultat «PR_8.1, PR_8.2 et PR_8.3 fausses» avec modèle du processus (MB) est interprété comme suit :

- Couplé à un comportement nominal du processus, le contrôleur ne peut pas produire la séquence de sorties qui permet d'assurer la preuve.

Pour ce type de propriétés, l'approche MB permet d'augmenter la portée de la preuve en incluant la détection de blocages dans le modèle du contrôleur dûs à son interaction avec le modèle de processus.

5.3.3 Remarques sur les deux approches : NMB et MB

Le tableau 5 dresse un bilan comparatif de l'intérêt de l'usage d'une approche «non model-based» par rapport à une approche «model-based». On peut dans un premier temps remarquer qu'aucune des deux approches ne prend l'ascendant sur l'autre. Chacune a des avantages et des point faibles, variables selon la nature des propriétés. Par exemple, l'approche NMB est bien adaptée à la preuve de propriétés de sûreté exprimées avec uniquement des variables contrôlables, tandis que l'approche MB donne des résultats plus satisfaisants pour les propriétés de vivacité.

On notera également de l'approche MB donne toujours des résultats significatifs même si parfois leur portée est plus limitée qu'avec l'approche NMB et si le temps de calcul est systématiquement beaucoup plus grand.

Tableau 5: Analyse comparative de l'intérêt de l'emploi des approches NMB et MB.

Propriété				Analyse des résultats expérimentaux	
Nom	Enoncé de la propriété	Type	Variables	Non Model-Based (NMB) : intérêt de l'approche dans l'absolu	Model-Based (MB) : intérêt de l'approche en relatif à l'approche NMB
PR_1	Les ordres envoyés au distributeur qui commande le vérin L1C ne doivent jamais être émis en même temps	S	C	fort	réduit
PR_2	Si l'ordre de descendre le vérin vertical est présent, alors on ne doit pas avoir l'émission d'ordres de commande relatifs aux mouvements des vérins horizontaux	S	C	fort	réduit
PR_3	S'il y a émission d'un ordre de commande correspondant aux mouvements des vérins horizontaux alors le capteur «vcu» doit être à «1»	S	C, O	aucun	majeur
PR_4	Après avoir été saisie, une pièce ne peut être lâchée qu'au droit de sa position de dépose	S	O	aucun	majeur
PR_5	Il n'y a de mouvements horizontaux que si le capteur «vcu» est à «1»	S	O, P	aucun	majeur
PR_6	Le modèle du contrôleur est toujours vivant	V	C	modéré	augmenté
PR_7	Si une pièce est détectée par pp1, pp2 ou pp3, alors on aura forcement, dans le futur, la sortie d'un des vérins horizontaux	V	O, P	aucun	majeur
PR_8	Si une pièce est détectée par pp1, pp2 ou pp3, alors on aura forcement, dans le futur, l'aspiration de la pièce	V	O	modéré	augmenté
PR_9	Quand le vérin vertical descend, tous les autres vérins sont et restent en fin de course	S	P	aucun	majeur

Type : S (sûreté), V (Vivacité) - Variables : C (Contrôlable), O (Observable), P (Processus)

5.3.4 Remarques sur la spécification du contrôleur de l'étude de cas

L'analyse des résultats de la vérification des propriétés sur notre étude de cas à mis en évidence des problèmes dans la spécification du contrôleur (donnée figure 52, "Spécification du contrôleur de l'exemple support.", page 69).

Si la propriété PR_2 est «*fausse*» en vérification NMB, c'est parce que le model-checker a identifié un contre-exemple comportant une situation du contrôleur pour laquelle les étapes 11 et 30 sont actives simultanément lorsqu'il n'y a pas de restriction sur les évolutions des variables «s1», «s2» et «s3». Pour éviter ce problème, la solution proposée consiste à utiliser des actions conditionnelles pour la sortie *VCGD* (associée aux étapes 30, 31, 35 et 37 du SFC 4). Les conditions ne devant être vraies que lorsque l'on a «$X11 + X21$ vrai», c'est-à-dire lorsque aucune des étapes qui émettent les sorties L1CGO et L2CGO n'est active.

Si la propriété PR_6 est «*fausse*» en vérification MB, c'est parce que le model-checker a identifié un contre-exemple comportant un blocage du contrôleur (deadlock) dans la situation {2, 10}. Après analyse des réceptivités associées aux transitions sensibilisées, on remarque que la transition 5 en aval de l'étape 2 reste non franchissable si une pièce est apparue dans la goulotte 1. En effet, la réceptivité de la transition 5 est trop restrictive. Seule l'absence de pièce dans la goulotte 2 est nécessaire à la désactivation de l'étape 2. Pour éviter ce problème, la solution proposée consiste à remplacer la réceptivité de la transition 5 par $pp2$ et celle de la transition 6 par $pp3$.

Si les propriétés PR_7.1 et PR_7.3 sont «*fausses*» en vérification MB, c'est parce que le model-checker a identifié un contre-exemple comportant le même blocage du contrôleur que pour la propriété PR_6 dans la situation {2, 10}. Les corrections apportées précédemment sur les réceptivités des transitions 5 et 6 doivent résoudre ce problème.

Finalement, la spécification corrigée du contrôleur est présentée figure 72, et les résultats de la vérification utilisant uniquement l'approche donnant les résultats les plus forts sont regroupés dans le tableau 6. Désormais toutes les propriétés sont vraies et, à titre indicatif, si l'on demande à NuSMV de les vérifier toutes ensembles (et non pas séparément) le temps de calcul global est de 561 min.

Tableau 6: Résultats finaux de la vérification du contrôleur corrigé

Propriété	Type de propriété	Type des variables	Approche retenue	Temps de calcul	Résultat de la preuve
PR_1	S	C	NMB	7 s	vraie
PR_2	S	C	NMB	8 s	vraie
PR_3	S	C, O	MB	108 min	vraie
PR_4	S	O	MB	108 min	vraie
PR_5	S	O, P	MB	109 min	vraie
PR_6	V	C	MB	348 min	vraie
PR_7.1	V	O, P	MB	162 min	vraie
PR_7.2	V	O, P	MB	146 min	vraie
PR_7.3	V	O, P	MB	160 min	vraie
PR_8.1	V	O	MB	150 min	vraie
PR_8.2	V	O	MB	160 min	vraie
PR_8.3	V	O	MB	156 min	vraie
PR_9	S	P	MB	109 min	vraie

Type : S (sûreté), V (Vivacité) - Variables : C (Contrôlable), O (Observable), P (Processus)

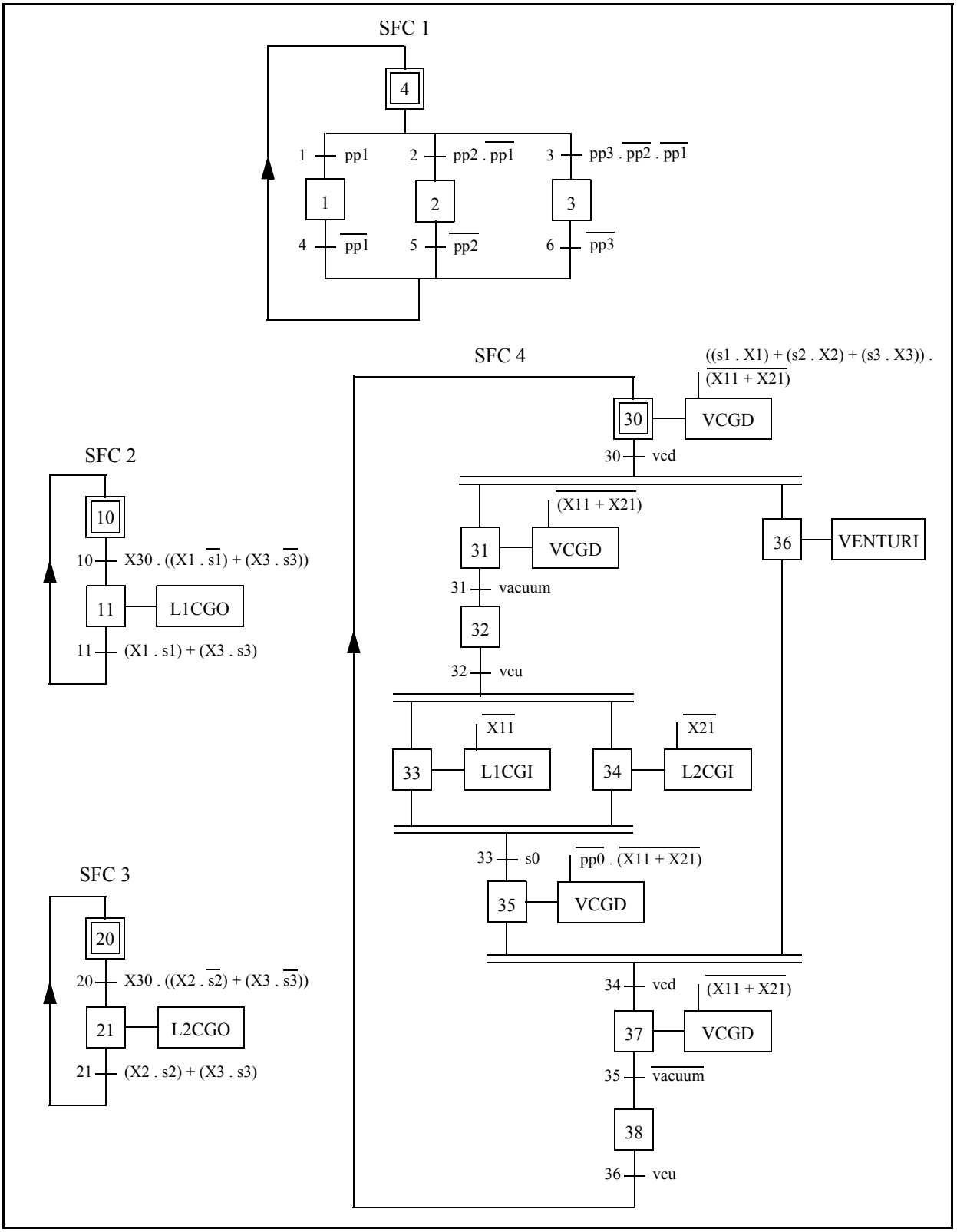


Figure 72: Spécification validée du contrôleur.

5.4 Proposition d'une approche mixte NMB-MB

L'analyse des résultats expérimentaux de notre étude de cas méritent une tentative de généralisation. Nous proposons ainsi une démarche de vérification formelle des SED par model-checking s'appuyant sur une utilisation conjointe de la vérification «Model-Based» et de la vérification «Non Model-Based». Notre objectif étant l'amélioration de la sûreté de fonctionnement des SED, chaque propriété sera traitée avec l'approche qui procure le résultat le plus fort, c'est à dire le plus sécuritif.

L'étude du tableau 6 montre que l'approche NMB donne des résultats plus forts pour les propriétés PR_1 et PR_2, tandis que l'approche MB est préférable pour toutes les autres propriétés. Le deux caractéristiques importantes qui permettent de sélectionner les propriétés pour une vérification par une approche ou par l'autre sont donc :

- le type de la propriété - sûreté ou vivacité,
- l'utilisation ou non de variables autres que les variables observables.

5.4.1 Propriétés de sûreté portant sur des variables contrôlables

Les propriétés de sûreté portant sur des variables contrôlables peuvent s'exprimer de la façon suivante : «une combinaison de la valeur de variables des sorties du contrôleur et de ses états internes n'apparaît jamais», ou sa contraposée. Pour garantir que le contrôleur est «robuste» vis-à-vis de toutes les variations possibles de ses entrées, il est ici préférable de ne pas faire d'hypothèse restrictive sur le processus et donc d'utiliser l'approche NMB.

5.4.2 Propriétés de sûreté portant sur au moins une variable non contrôlable

Les propriétés de sûreté portant sur des variables observables peuvent s'exprimer de la façon suivante : «une combinaison de la valeur de variables dont une au moins est non contrôlable n'apparaît jamais», ou sa contraposée. Pour garantir que le contrôleur est «robuste» vis-à-vis de toutes les variations possibles de ses entrées, il serait ici également préférable de ne pas faire d'hypothèse restrictive sur le processus, mais le contrôleur n'ayant pas le contrôle direct d'une au moins des variables il faudra compter sur le comportement du processus. Il est donc indispensable d'ajouter un comportement du processus pour que la preuve puisse être conduite. Cependant, plus le modèle sera permissif (dans le sens d'une faible restriction de l'évolution des entrées du contrôleur), plus la portée de la preuve sera grande. L'approche MB doit donc être retenue, moyennant cette remarque.

5.4.3 Propriétés de vivacité

Les propriétés de vivacité s'expriment de la façon suivante : «il existe une séquence d'évolution de la valeur des variables qui permet d'atteindre un état donné», ou «il existe toujours une séquence d'évolution de la valeur des variables qui permet d'atteindre un état donné». Contrairement aux propriétés de sûreté pour lesquelles on cherchait à montrer que, malgré une grande liberté de variation des variables d'entrées du contrôleur, le système contrôlé n'atteindra jamais certains états, ici on veut prouver que malgré les restrictions d'évolution créées par le processus, un chemin existe pour atteindre un état. Plus le modèle sera restrictif (dans le sens d'une faible liberté sur l'évolution des entrées du contrôleur), et plus la portée de la preuve sera grande. Nous retenons donc l'approche MB pour les propriétés de vivacité.

5.4.4 Arbre de décision selon la propriété

La figure tableau 73 présente sous la forme d'un arbre de décision l'approche mixte de vérification proposée.

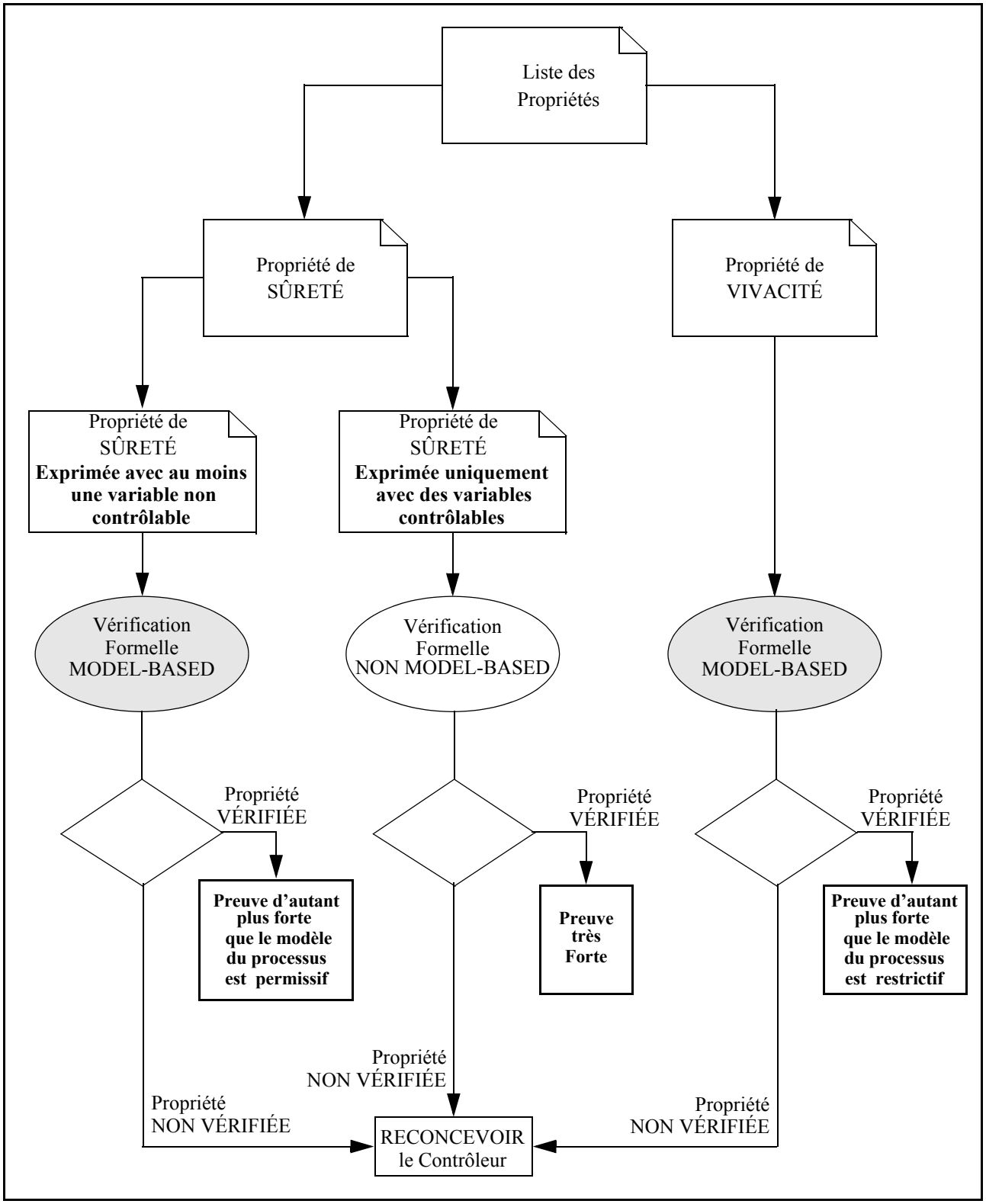


Figure 73: Proposition d'une approche de vérification mixte NMB-MB.

5.5 Bilan du chapitre 5

Dans ce chapitre, nous avons tout d'abord montré comment traduire de façon systématique un modèle prêt à être vérifié en langage NuSMV. Cette technique de traduction une fois appliquée à notre exemple support, toutes les propriétés élaborées précédemment ont été vérifiées avec les deux approches : MB et NMB (lorsque cela avait un sens).

Les résultats de vérification procurés par ces deux approches ont été comparés et discutés pour chacune des propriétés. La vérification MB nécessite systématiquement un effort de calcul plus important que la vérification NMB. Au contraire, les résultats des vérifications MB sont toujours utiles au concepteur alors que ceux obtenus en vérification NMB ne sont pas toujours significatifs. Enfin, nous avons mis en évidence que ces deux approches sont complémentaires et que, globalement, l'approche NMB donne des résultats plus forts pour les propriétés de sûreté et l'approche MB donne des résultats plus forts pour les propriétés de vivacité.

Finalement, considérant que l'exemple traité et les propriétés vérifiées étaient suffisamment représentatifs des SED, nous avons tenté une généralisation de ces résultats et présenté une approche proposant un couplage des vérifications NMB et MB. Nous avons pour cela privilégié la sûreté de fonctionnement au temps de calcul. Cette approche est basée sur une double typologie des propriétés et des variables utilisées pour les exprimer. On distingue ainsi les propriétés de vivacité (devant systématiquement être vérifiées par une approche MB) des propriétés de sûreté qui, selon qu'elles sont exprimées uniquement avec des variables contrôlables ou non doivent être vérifiées avec une approche NMB ou MB. Cette approche couplée propose un cadre de vérification plus rigoureux et plus sûr que celui de l'utilisation, la plus souvent pratiquée d'une seule des approches NMB ou MB.

Conclusions et perspectives

L'objectif principal de cette thèse était, en conduisant une étude comparative, d'évaluer les apports, les limites et les complémentarités de la vérification des SED par model-checking en prenant en compte, ou non, un modèle du processus commandé. Une telle étude ne pouvait être conduite sans le traitement d'une ou de plusieurs études de cas. En effet, comment comparer dans l'absolu la pertinence des résultats de vérifications pratiquées sur des systèmes aux frontières d'isolement différentes ? Nous avons donc adopté une démarche inductive, partant du traitement d'un cas représentatif et qui conduise à la généralisation des conclusions obtenues. Dans la pratique, plusieurs exemples ont été traités durant ce travail de thèse. Le traitement de ces études de cas a permis de mettre au point le cahier des charges de l'exemple que nous avons développé, qui est de notre point de vue à la fois de taille compatible avec les exigences de la rédaction d'un mémoire et porteur des comportements génériques et des difficultés classiquement rencontrés en modélisation et en vérification des SED logiques non temporisés. De plus, le traitement de ces exemples nous a permis de choisir un nombre limité de propriétés à vérifier, représentatives des différents types de propriétés que cherche à prouver l'automaticien et des difficultés à les formaliser.

La prise en compte d'un modèle de processus en vérification formelle du comportement d'un contrôleur pose naturellement le problème de la construction de ce modèle de processus. La plupart des auteurs qui se sont réellement confrontés à ce problème émettent en commun les deux conclusions suivantes : les techniques modulaires de construction sont indispensables au traitement des cas de taille significative ; l'obtention du modèle complet du processus par composition (synchrone ou asynchrone) des modèles des composants élémentaires génère un modèle de taille importante qui de plus traduit des comportements sans signification physique. Ces deux objectifs (privilégier une grande modularité de la construction et éviter la composition d'automates) nous ont conduit à proposer une méthode originale de construction du modèle de processus. Dans notre approche, la modularité est garantie par la décomposition du processus en chaînes fonctionnelles. Le modèle complet est ensuite obtenu par coordination des comportements des modules élémentaires, grâce à un séquenceur qui permet une évolution asynchrone des modules jusqu'à atteindre une situation stable du processus qui traduit l'aboutissement de sa réaction aux ordres émis par le contrôleur.

Une fois les modèles du contrôleur et du processus établis, il reste à les coupler pour traduire leurs interactions. Là encore, il convient d'éviter la composition. Nous avons pour notre part considéré un SED comme étant composé d'un contrôleur et d'un processus en boucle fermée, tous deux aux comportements asynchrones, synchronisés lors des échanges d'informations (cadencés par les phases de

lecture des entrées et d'affectation des sorties du contrôleur). Cette interaction comportementale se traduit alors aisément par une synchronisation de type «appel-réponse» dans les automates de comportement du contrôleur et du processus. Elle est réalisée grâce à un séquenceur général qui permet de plus la vérification des propriétés aux instants propices : en fin d'évolution du contrôleur et en fin d'évolution du processus.

Enfin, rappelons que pour mener à bien nos travaux, nous avons dû définir une classe d'automates spécifique au fort pouvoir d'expression. Il s'agit d'automates non déterministes, communiquant par partage de variables logiques et aux transitions labellées par des expressions Booléennes. Ces mêmes automates sont utilisés, conjointement à une logique temporelle (LTL ou CTL), pour formaliser des propriétés complexes traduisant des comportements séquentiels attendus ou rejetés.

Les résultats opérationnels obtenus sont encourageants en ce qui concerne les temps de calcul et la taille mémoire nécessaire au model-checking (moins que 10 heures pour la preuve des 29 propriétés du tableau 7 - Annexe 1, avec une machine équipée d'un processeur Pentium III à 1 GHz et de 1 Go de mémoire vive, à l'aide de la version 2.1.2 du model-checker NuSMV).

De nombreuses perspectives s'offrent à nous pour prolonger ces travaux. En tout premier lieu, signalons que l'automatisation de la construction du modèle de processus à partir d'une bibliothèque de modèles comportementaux des modules fonctionnels est en cours de réalisation. Dans ce mémoire nous nous sommes attachés à montrer qu'une telle construction, ainsi que la génération du séquenceur de processus, était systématique. Un démonstrateur est actuellement en cours de réalisation pour implémenter ces mécanismes.

A moyen terme, nous désirons étendre notre approche aux SED temporisés. Un projet de recherche a été soumis dans ce sens, et approuvé par la «Fundação para a Ciência e Tecnologia» (FCT). Il implique la collaboration étroite entre les Départements d'Ingénierie Mécanique et d'Informatique de l'École d'Ingénierie de l'Université du Minho et le LURPA.

Enfin, rappelons que tous nos modèles traduisent des comportements du contrôleur et du processus en état de bon fonctionnement. Notre projet à plus long terme est d'utiliser le model-cheking pour valider le comportement du contrôleur et du processus sur occurrences de fautes.

Références bibliographiques

[ALEXANDRE & BOUDAUD 1997] Jeandel Alexandre, and Fabrice Boudaud. *Physical system modelling languages: from ALLAN to Modelica*. In Building Simulation '97, IBPSA conference, http://www.modelica.org/papers/p303.pdf, 1997. http://citeseer.ist.psu.edu/alexandre97physical.html

[ALPAN & JAFARI 2002] Gülgün Alpan, Mohsen A. Jafari *Synthesis of a Closed-Loop Combined Plant and Controller Model* IEEE Transactions on systems, man and cybernetics - Part B: Cybernetics, Vol. 32, Nº2, April 2002.

[ALPERN & SCHNEIDER 1985] B. Alpern, F. B. Schneider : *Defining liveness*. Inf. Process. Lett. 21, 4 (Oct.), 181–185; 1985.

[ALUR *et al.* 1993] R. Alur, C. Courcoubetis, T. A. Henzinger, Pei-Hsin Ho : *Hybrid automata: an algorithmic approach to the specification and analysis of hybrid systems*, In Workshop on theory of Hybrid systems, Lyngby, Denmark, October 1993. LNCS 736, Spinger Verlag.

[ALUR & DILL 1994] R. Alur, D. L. Dill : *Automata for Modelling Real-Time Systems*, Theoretical Computer Science, 126(2) : 183-236, April 2004.

[ALUR & HENZINGER 1991] Alur R., and Henzinger T.A. : *Logics and models of real time: a survey*, REX Workshop on Real-Time: Theory in practice (Lecture Notes in Computer Science n°600), pp. 74-106, 1991.

[AMERONGEN 2003] Job Van Amerongen : *Mechatronic Design*, Mechatronics 13 (2003), 1045–1066.

[BALEMI *et al.* 1993] 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, p. 1040-1059.

[BARESI *et al.* 1998] Luciano Baresi, Stefania Carmeli, Antonello Monti, and Mauro Pezzè : *PLC Programming Languages: A formal Approach* Automation 98. ANIPLA. November 1998.

[BARESI *et al.* 2000] Luciano Baresi, Marco Mauri, Antonello Monti, and Mauro Pezzè : *PLC-TOOLS: Design, Formal Validation, and Code Generation for Programmable Controllers,* In Formal methods in PLC programming Special Session at IEEE Conference on Systems, Man, and Cybernetics Nashville (USA), October 2000.

[BARTON 1992] Paul Inigo Barton : *The Modelling and Simulation of Combined Discrete/Continuous Processes*, Ph.D. Thesis, University of London, 1992.

[BIEREL *et al.* 1997] Evelyne Bierel, Olivier Douchin, Pascal Lhoste, "*Grafcet: From the theory to implementation*", European Journal of Automation, (31)3, 1997.

[BORNOT *et al.* 2000] Sébastien Bornot, Ralf Huuck, Ben Lukoschus: *Verification of Sequential Function Charts Using SMV,* At the PDPTA 2000 special session on Formal Validation, Las Vegas, Nevada, USA, June 26-29, 2000

[BRAND & ZAFIROPULO 1983] D. Brand, P. Zafiropulo : *On communicating finite state machines*, JACM, 30(2) : 323-342.

[BRYANT 1986] R. E. Bryant : *Symbolic boolen manipulation with Ordered Binary Decision Diagrams*, ACM Computing surveys, 24(3) : 293-318, 1992.

[BROWE *et al.* 1986] M C Browne, E M Clarke, D L Dill and B Mishra : *Automatic verification of sequential circuits using temporal logic*, IEEE Transactions on Computers, C-35(12) 1035-1044 , 1986.

[BURCH *et al.* 1992] J. R. Burch, E M Clarke, K. L. McMillan, D. L. Dill and L. J. Hwang : *Symbolic Model-Checking : 1020 States and Beyond*, Information and Computing, 98(2) : 142-170, 1992.

[CANET *et al.* 2000] G. Canet, S. Couffin, J.-J. Lesage, A. Petit, P. Schnoebelen : *Towards automatic verification of PLC programs written in Instruction List*, Proceedings of the IEEE, SMC, 2000.

[CARRÉ-MÉNÉTRIER *et al.* 2002] V. Carré-Ménétrier, A. Philippot, A. Tajer, F. Gellot : *Démarche de synthèse pour la commande d'un préhenseur pneumatique*, Présentation dans le Groupe de travail COSED, Septembre 2002.

[CHEUNG & KRAMER 1999] S. C. Cheung, J. Kramer : *Checking Safety Properties Using Compositional Reachability Analysis*, ACM Transactions on Software Engineering and Methodology, Vol.8, No. 1, January 1999.

[CLARKE *et al.* 1986] E. M. Clarke, E.A. Emerson, A.P. Sistla : *Automatic verification of finite-state concurrent systems using temporal logic specifications*. In ACM Transactions on Programmation Languages and Systems, 8(2) : 244 - 263, 1986.

[CLARKE *et al.* 1999] E. M. Clarke, O. Grumberg, D. A. Peled : *Model-Checking*. The MIT Press, 1999.

[CORBETT & AVRUNIN 1995] J.C. Corbett, G. S. Avrunin : *Using integer programming to verify general safety and liveness properties*. Formal Methods Syst. Des. 6, 1 (Jan.), 97–123.; 1995.

[CORTÉS *et al.* 2003] Luís Alejandro Cortés, Petru Eles, Zebo Peng : *Modeling and formal verification of embedded systems based on a Petri net representation*, Journal of Systems Architecture 49 (2003) 571–598.

[COUDERT *et al.* 1990] O. Coudert, C. Berthet, J.-C. Madre : *Verification os Synchronous Sequential Machines Based on Symbolic Expressions*. In Proc. Int. Workshop of Automatic Verification Methods for Finite State Systems (CAV'89), Grenoble June 1989, Volume 407 de «Lecture Notes in Computer Science», pages 365-373, Springer, 1990.

[DAVIS & HAMSCHER 1988] R. Davis, W. Hamscher : *Model-based reasoning : Trouble shooting*. In exploring Artificial Intelligence : Survey Talks from the National Conferences on Artificial Intelligence, Schrobe H. (dir), Morgan Kaufmann, pp. 297-346, 1988.

[DE SMET *et al.* 1999] O. de Smet, J.-M. Roussel, N. Hévin : *Identification de machine séquentielle binaire : application à un système réactif*. Proceedings of the 2nd Congress on modelisation of réactive systems (MSR'99), pp. 351-360, Cachan, France, March 1999.

[DE SMET *et al.* 2000] O. de Smet, S. Couffin, O. Rossi, G. Ganet, J.-J. Lesage, Ph. Schnoebelen, H. Papini : *Safe programming of PLC using formal verification methods*, Proceedings of the ICP Conference, October 2000.

[DE SMET & ROSSI 2002] Olivier de Smet and Olivier Rossi, *Verification of a controller for a flexible manufacturing line written in Ladder Diagram via model-checking*, 21th American Control Conference, ACC'02, Anchorage (USA), CDROM paper nº734, pp. 4147-4152, May 2002.

[DWYER & CLARKE 1994] M. B. Dwyer & L. A. Clarke : Dataflow analysis for verifying properties of concurrent programs. In Proceedings of the 2nd ACM SIGSOFT Symposium on the Foundations of Software Engineering (SIGSOFT'94, NewOrleans, LA, Dec.) .ACM Press, New York, NY, 62–75; 1994.

[ELMQVIST *et al.* 1999] Hilding Elmqvist, Sven Erik Mattsson, Martin Otter: *Object-Oriented and Hybrid Modeling in Modelica*. Journal Européen des systèmes automatisés, 35,1/ 2001.

[EMERSON & PERN 1986] E. A. Emerson and J. Y. Alpern : *«Sometimes» and «Nor Never» Revisited; On branching versus Linear Time Temporal Logic*. Journal of the ACM 33(1) : 151-178, 1986.

[EN 2002] European Standard 60848: *GRAFCET specification language for sequential function charts*, 2002.

[FREY & LITZ 2000-a] Georg Frey and Lothar Litz : *Formal methods in PLC programming*, Proceedings of the IEEE Conference on Systems, Man and Cybernetics, SMC 2000, Nashville, October 8-11, 2000.

[FREY & LITZ 2000-b] Georg Frey and Lothar Litz : *Correctness analysis of Petri Net Based Logic Controllers*. Proceedings of the American Control Conference, ACC2000, Chicago, 2000.

[FUJINO *et al.* 2000] Kazuhisa Fujino, Kei Imafuku, Yuh Yamashita, Hirokazu Nishitani : *Design and verification of the SFC program for sequential Control*, Computers and Chemical Engineering 24 (2000) 303-308.

[GOURCUFF 2004] V. Gourcuff : *Verification of a timed multitask system with UPPAAL*, Mémoire de DEA, LURPA, ENS de Cachan, 2004.

[GOUYON 2001] David Gouyon *Application de techniques de sysnthèse de la commande en ingéniérie d'automatisation*, Diplôme d'études approfondies; Université Henri Poincaré, Nancy I; 2001.

[GOUYON *et al.* 2003] David Gouyon, Jean-François Pétin et Alexia Gouin : *Modèles de procédé et de ses spécifications pour la synthèse de la commande*, MSR'2003, Metz, 6-8 octobre 2003, pp. 45-60

[GOUYON 2004] David Gouyon *Techniques de sysnthèse de la commande en ingéniérie d'automatisation*, Thése de doctorat; Université Henri Poincaré, Nancy I; 2004.

[HASSAPIS *et al.* 1998] G. Hassapis, I. Kotini, and Z. Doulgeri : *Validation of a control system software specified in SFC notation by using hybrid automata*, INCOM'98, Vol.II, pp. 65-70, 1998.

[IEC 1993] IEC (International Electrotechnical Commission): *IEC Standard 61 131-3: Programmable Controlers - Part 3*, 1993.

[LIU, 2004] Yongqian Liu : *Modélisation et simulation pour l'automatisation orientée performances de systèmes de production*, Rapport de Recherche Post Doctorale de l'Université Henri Poincaré, Nancy 1, séance 6 janvier 2004, pp 11-27.

[KOWALEWSKI & PREUßIG 1996] Stefan Kowalewski and Jörg Preußig : *Verification of sequential controllers with timing functions for chemical processes*, 13th IFAC World Congress, San Francisco, USA, 30 June - 5 July, 1996, invited session on Advances in Discrete Event Control.

[KLEIN & 2004] Stéphane Klein : isttable; *Fault detection of discrete event systems using an identification approach*, Thése de doctorat, Université de Kaiserslautern, juin 2005.

[KUMAR 1991] R. Kumar, *Supervisory Synthesis Techniques for Discrete Event Dynamical systems*, Thesis for Ph. D. Degree, University of Texas, August 1991

[LAFORTUNE *et al.* 2001] S. Lafortune, D. Teneketzis, M. Sampath, R. Sengupta, K. Sinnamohideen : *Failure Diagnosis of Dynamic Systems : An approache based on Discrete Event Systems*, Proceedings of the American Control Conference ACC'2001, pp. 2058-2071, Arlington, VA (USA), June 25-27, 2001.

[LAMPÉRIÈRE-COUFFIN *et al.* 1999] S. Lampérière-Couffin, O. Rossi, J.-M. Roussel and J.-J. Lesage : *Formal validation of PLC programs: A survey*, Proceedings of the ECC'99, Karlsruhe Germany, August 1999, cdrom paper N° 741, 6 pages.

[LAMPÉRIÈRE-COUFFIN & LESAGE 2000] S. Lampérière-Couffin and J.-J. Lesage : *Formal Verification of the Sequential Part of PLC Programs*, Proceedings of The 5th Workshop on Discrete Event Systems (WODES'2000), Ghent, Belgium, August 2000.

[LEBRUN 2003] Michel Lebrun : *Simulation et CAO en automatique et mécatronique*, Techniques de l'Ingénieur, traité Informatique industrielle, S 7 260, 2003.

[LEE & LEE 1998] Bilung Lee and Edward A. Lee, *Hierarchical Concurrent Finite State Machines in Ptolemy* {bilung, eal}@eecs.berkeley.edu, University of California at Berkeley, Proceeding of International Conference on Application of Concurrency to System Design, p. 34-40, Fukushima, Japan, March 1998.

[LI & WONHAM 1988] Yong Li and W. M. Wonham : *Controllability and Observability in the state feed-back Control of Discrete Event Systems*. Proceedings of the 27th Conference on Decision and Control. Austin, Texas. December 1988. pp. 203-208.

[MANNA & PNUELI 1995] Z. Manna, A. Pnueli : *Temporal Verification of Reactive Systems : Safety.* Springer-Verlag, NewYork, NY.; 1995.

[MACHADO *et al.* 2003-a] José M. Machado, Bruno Denis, Jean-Jacques Lesage, Jean-Marc Faure, Jaime C. L. Ferreira da Silva : *Increasing the efficiency of PLC Program Verification using a plant model*, International Conference on Industrial Engineering and Production Management (IEPM'2003), Porto, Portugal, 26-28th May 2003.

[MACHADO *et al.* 2003-b] José M. Machado, Bruno Denis, Jean-Jacques Lesage, Jean-Marc Faure, Jaime C. L. Ferreira da Silva : *Model of Mechanism behavior for verification of PLC programs*, 17th Edition of the International Congress of Mechanical Engineering (COBEM'2003), São Paulo, Brazil, 10-14th November 2003.

[MALER *et al.* 1992] O. Maler, Z. Manna, and A. Pnueli : *From Timed to Hybrid Systems*. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, Proceedings of the REX Workshop ``Real-Time: Theory in Practice", volume 600 of Lecture Notes in Computer Science, pages 447-484. Springer-Verlag, 1992.

[MARCÉ & LE PARC 1993] Lionel Marcé and P. Le Parc : "*Defining the semantics of languages for programmable Controllers with synchronous processes*", Control Engineering Practice, Vol.1, Nº1, pp. 79-84, 1993.

[MATTSSON *et al.* 1998] Sven Erik Mattsson, Hilding Helmqvist, Martin Otter : *Physical System Modelling with Modelica*, Control Engineering Practice 6 (1998), 501-510.

[MEDA-CAMPAÑA & LÓPEZ-MELLADO 2002] M.E. Meda-Campaña and E. López-Mellado, *Incremental synthesis of Petri net models for identification of discrete event systems*, Proceedings of the 41st IEEE Conference on Decision and Control, 805-810., December 2002, Las vegas, Nevada USA.

[MERTKE & FREY 2001] T. Mertke and G. Frey: *Formal Verification of PLC-programs generated from signal interpreted Petri Nets*, Proceedings of the SMC 2001, Tucson (AZ), pp. 2700-2705, Oct 2001.

[MERTKE & MENZEL 2000] T. Mertke and T. Menzel : *Methods and tools to the verification of safety-related control software*, Proceedings of the IEEE SMC, 2000.

[MOODY & ANTSAKLIS 1998] J.O. Moody and P.J. Antsaklis. *Petri net supervisors for DES with uncontrollable and unobservable transitions*. IEEE Transactions on Automatic Control, 1999. To appear. http://citeseer.ist.psu.edu/article/moody98petri.html

[NDJAB HAGBEBELL 1999] Constant Ndjab Hagbebell *Synthèse de la commande des systèmes à événements discrets par Grafcet*, Thèse de doctorat, Université de Reims Champagne Ardenne, 1999.

[OKANO *et al.* 1999] Kozo Okano, Satoshi Hattori, Akira Yamamoto, Teruo Higashino and Kenichi Tanigugi: *Specification of real-time systems using a timed automata model with shared variables and verification of partial-deadlock freeness*, Proceedings of the 1999 ICPP workshops, pp. 576-581 (Sep. 1999).

[OLENDER & OSTERWEIL 1990] K. M. Olender, L. J. Osterweil : *Cecil : A sequencing constraint language for automatic static analysis generation*. IEEE Trans. Softw. Eng. 16, 3 (Mer.), 268–280.; 1990.

[PAPADOPOULOS & MCDERMID 2001] Y. Papadopoulos, J. McDermid : *Automated Safety Monitoring : A Review and Classification of Methods*, International Journal of condition monitoring and Diagnostic Engineering Management. 4(4): 14/32, ISSN 1362-7681, October 2001.

[PHILIPPOT *et al.* 2003] A. Philippot, A. Tajer, F. Gellot, V. Carré-Ménétrier : *Synthèse de la commande spécifiée en Grafcet : application à un préhenseur pneumatique*, MSR'2003, Metz, 6-8 octobre 2003, pp. 61-75.

[PHILIPPOT *et al.* 2004] A. Philippot, A. Tajer, F. Gellot, V. Carré-Ménétrier : *On-line synthesis approach based on structured plant modelling*, Proceedings of the 7th Workshop on Discrete Event Systems (WODES'2000), Reims, France, September 2004.

[PIXLEY 1992] C. Pixley : A Theory and Implementation of Sequential Hardware Equivalence. IEEE Transactions on Computer Aided Design of Integrated Circuits, 11(12) : 1469-1478, 1992.

[PROBST *et al.* 1997] S. Probst, G. Powers, D. Long, I. Moon : *Verification of a logically controlled, solids transport system using symbolic model-checking*. Computers & Chemical Engineering (UK), vol. 21, nº 4, Elsevier 1997, pp. 417-429.

[RAMADGE & WONHAM 1987] P. J. Ramadge and W. M. Wonham *Spervisory control of a class of discrete event processes* SIAM J. Control Optimization, 25(1), 1987, p. 206-230.

[RAMADGE & WONHAM 1989] P. J. Ramadge and W. M. Wonham *The control of discrete event systems*, Proceedings of IEEE 77(1), pp. 81-98, 1989.

[RAUSCH & KROGH 1998] M. Rausch and B. H. Krogh : *Formal Verication of PLC Programs*, American Control Conference, Philadelphia, PA, USA, June 1998.

[REITER 1987] R. Reiter : *A Theory of diagnosis from First Principles*, Artificial Intelligence 32, pp. 57-96, 1987.

[ROSSI 2004] Olivier Rossi : *Validation formelle de programmes Ladder Diagram pour Automates Programmables industriels*, Thése de doctorat, École Normale Supérieure de Cachan, juin 2004.

[ROUSSEL & DENIS 2002] J-M. Roussel, B. Denis : *Safety properties verification of ladder diagram programs* : Journal Européen des Systèmes Automatisés, Vol. 36, pp. 905-917. 2002.

[SCHNOEBELEN *et al.* 1999] Ph. Schnoebelen, B. Bérard, M. Bidoit, F. Laroussinie, A. Petit and others : *Vérification de Logiciels : Tcnhiques et outis du model-checking*, Vuibert 1999.

[SZTIPANOVITS & MISRA 1994] J. Sztipanovits, A. Misra : *Diagnosis of discrete event systems using ordered binary decision diagrams*. Proceedings of the 7th international Workshop on Principles of Diagnoses (DX'96), Val Morin (Canada), 1996.

[VAREA *et al.* 2002] Mauricio Varea, Bashir M. Al-Hashimi, Luis A. Cortés, Petru Eles and Zebo Peng; *Symbolic Model Checking of Dual Transition Petri Nets*; in Proceedings of the 10th International Symposium on Hardware/Software Codesign (CODES); pp. 43-48; Colorado, United States; 6-8th May 2002.

[VULCAIN 1998] Project VULCAIN (Validation par L'utilisateur de Logiciels de Commande d'Automatismes INdustriels); Partenariat entre les laboratoires de recherche LURPA et LSV de l'ENS de Cachan et la société ALCATEL.

[WING *et al.* 1997] Jeannette M. Wing and Mandana Vaziri-Farahani : *A case study in model-checking software systems*, Science of Computer Programming, vol. 28, 1997, pp. 273-299.

[WONHAM & RAMADGE 1987] W. M. Wonham, P. J. Ramadge *On the supremal controllable sublanguage of a given language*, SIAM J. Control Optimization, 25(3), 1987, pp. 637-659.

[ZAYTOON & CARRÉ-MÉNÉTRIER 1999] Janan Zaytoon and Véronique Carré-Ménétrier : *Grafcet et graphe d'états : comportement, raffinement, vérification et validation*, APII - JESA. Volume n°7/1999, pages 751 à 782.

[ZAYTOON *et al.* 1999] Janan Zaytoon, Constant Ndjab Hagbebell, Véronique Carré-Ménétrier : *Grafcet et graphes d'états : synthèse hors ligne de la commande,* APII – JESA. Volume 33 - n°7/1999, pages 783 à 814.

Références techniques

[AS 2004] Famic Technologies inc : *Automation Studio 5.0*, http://www.automationstudio.com/, 2004.

[FS 2004] Festo Software : *FluidSIM® 3.6*, http://www.festo.com, 2004.

[NuSMV 2002] Formal Methods group in the Automated Reasoning Systems, division at ITC-IRST, the Model Checking group at Carnegie Mellon University, the Mechanized Reasoning Group at University of Trento : *NuSMV 2.1.2 (compiled 2002-11-22) : NuSMV: a new symbolic model checker* http://nusmv.irst.itc.it/, 2002.

[UPPAAL 2004] Logiciel UPPAAL : UPPAAL 3.4.7, http://www.uppaal.com/, 2004.

Annexe 1
Modèle complet de l'exemple support utilisé

Nous présentons, dans cet annexe, le modèle complet du système utilisé comme exemple support.

Nous commencons par la présentation de la partie générique de la structure du modèle pour les deux approches Non Model-Based (NMB) et Model-Based (MB), puis nos présentons le modèle complet de notre exemple support pour l'approche MB.

A1.1 Structure générique des modèles

Pour l'approche MB le modèle est composé de (voir figure 74) :

- la valeur initiale des variables du modèle,
- le séquenceur général
- le modèle du contrôleur (le séquenceur du contrôleur enrichi de la loi de commande),
- le modèle complet du processus (séquenceur et modules) et
- le modèle complet des propriétés (séquenceur, modules, formule de logique temporelle).

Pour l'approche NMB le modèle est composé de (voir figure 75) :

- la valeur initiale des variables du modèle,
- le séquenceur général
- le modèle du contrôleur (le séquenceur du contrôleur enrichi de la loi de commande),
- le modèle «vide» du processus (séquenceur uniquement) et
- le modèle complet des propriétés (séquenceur, modules, formule de logique temporelle).

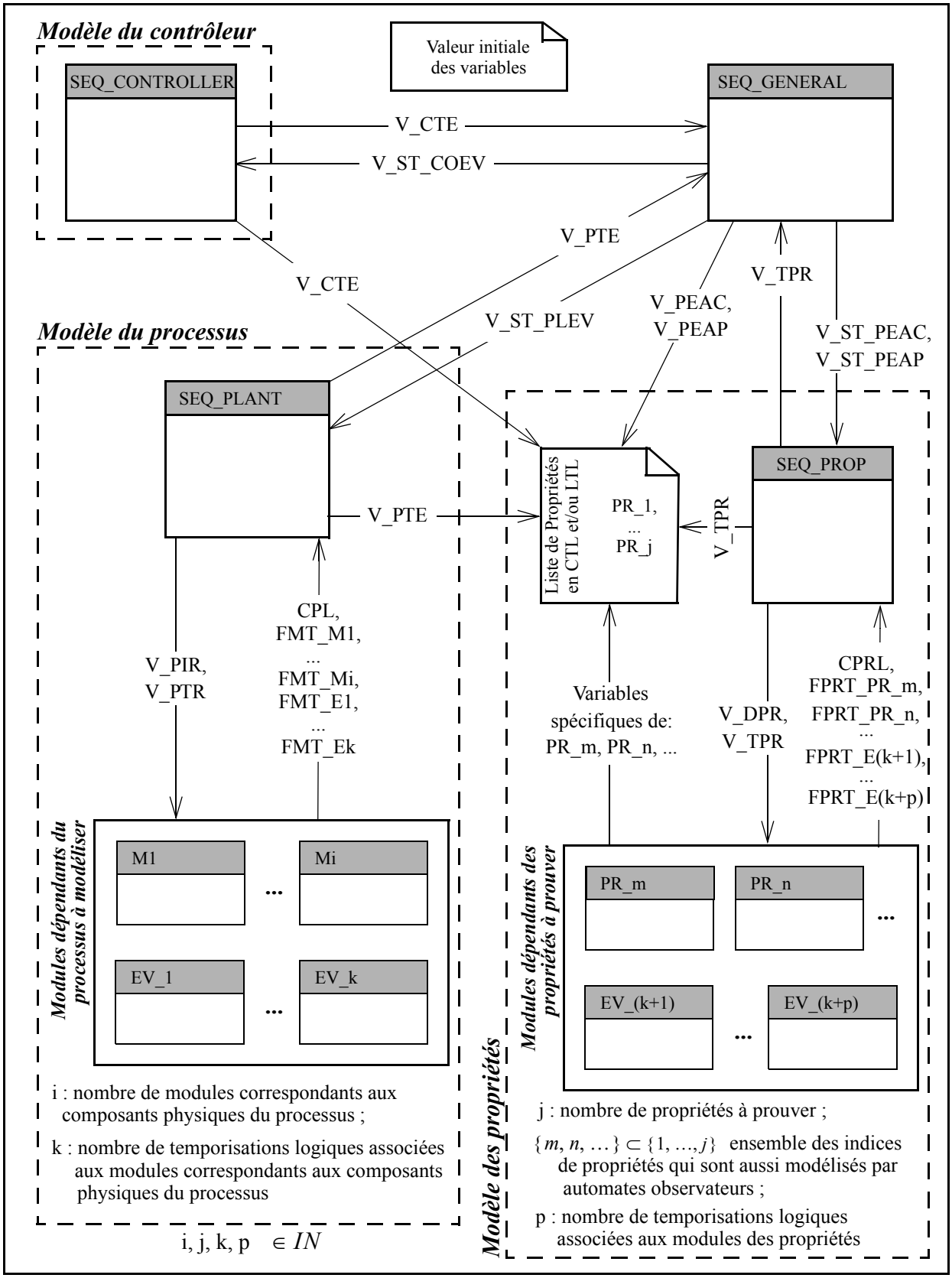


Figure 74: Structure génrique proposée pour un modèle selon approche MB.

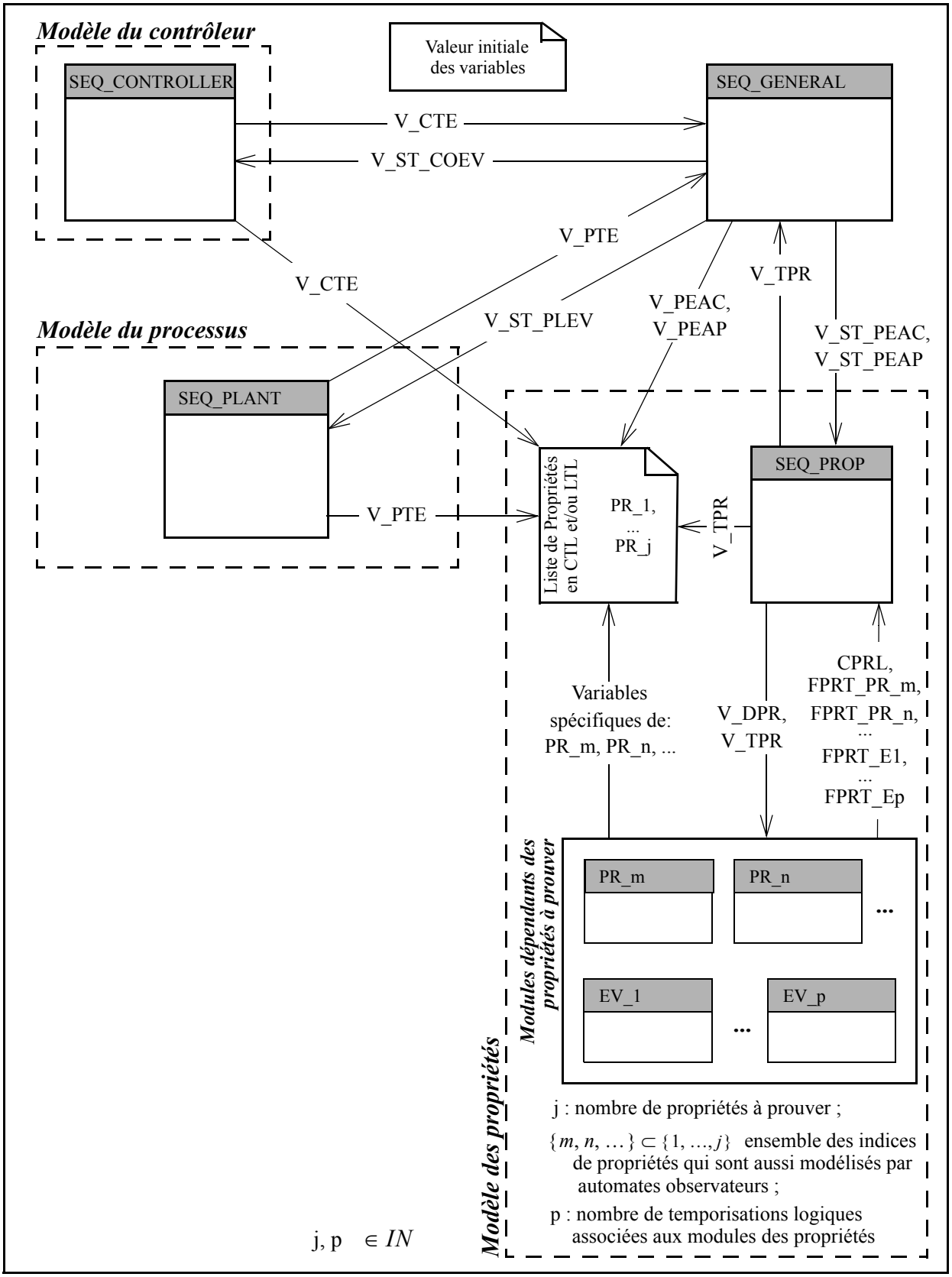


Figure 75: Structure génrique proposée pour un modèle selon approche MB

A1.2 Modèle complet pour l'approche model-based

Le modèle complet ici présenté est le modèle obtenu après correction du contrôleur dont les résultats de preuve sont présentés dans le Tableau 6, page 102. On utilise la structure générique de l'approche MB.

A1.2.1 Valeur initiale des variables du modèle

Valeur initiale des variables du modèle :

V_ST_COEV=0,
V_ST_PEAC=0,
V_ST_PLEV=0,
V_ST_PEAP=0,
V_PEAC=0,
V_PEAP=0,
V_CIR=0,
V_CFR=0,
V_CTR=0,
V_COE=0,
V_COW=0,
V_CEV=0,
V_CTE=1,
V_PIR=0,
V_PTR=0,
V_POU=0,
V_PTE=1,
V_SPR=0,
V_DPR=0,
V_TPR=1,
vcu=1,
vcd=0,
s0=1,
s1=0,
s2=0,
s3=0,
pp0=0,
pp1=0,
pp2=0,
pp3=0,
vacuum=0,
vcu_c=0,
vcd_c=0,
s0_c=0,
s1_c=0,
s2_c=0,
s3_c=0,
pp0_c=0,
pp1_c=0,
pp2_c=0,
pp3_c=0,
vacuum_c=0,
CF1=0,
CF2=0,
CF3=0,
CF4=0,
CF5=0,
CF6=0,
CF10=0,
CF11=0,
CF20=0,
CF21=0,
CF30=0,
CF31=0,
CF32=0,
CF33=0,
CF34=0,
CF35=0,
CF36=0,
X1=0,
X2=0,
X3=0,
X4=1,
X10=1,
X11=0,
X20=1,
X21=0,
X30=1,
X31=0,
X32=0,
X33=0,
X34=0,
X35=0,
X36=0,
X37=0,
X38=0,
VCGD=0,
L1CGO=0,
L1CGI=0,
L2CGO=0,
L2CGI=0,
VENTURI=0,
VCGD_c=0,
L1CGO_c=0,
L1CGI_c=0,
L2CGO_c=0,
2CGI_c=0,
VENTURI_c=0,
V_P2=0,
V_P3=0,
V_P4=0,
V_P5=1,
V_P6=0,
V_P7=0,
V_P8=0,
V_P9=1,
V_P10=0,
V_P11=0,
V_P12=0,
V_P21=1,
V_P22=0,
V_P23=1,
V_P24=0,
V_P25=0,
V_P26=1,
V_P27=0,
V_P28=0,
V_P29=1,
V_P30=0,
V_P31=0,
V_P32=1,
V_P33=0,
V_P34=0,
FMT_L1C=0,
FMT_L2C=0,
FMT_VC=0,
FMT_s0p=0,
FMT_s1p=0,
FMT_s2p=0,
FMT_s3p=0,
FMT_ZV=0,
FMT_Z0=0,
FMT_Z1=0,
FMT_Z2=0,
FMT_Z3=0,
CPL=0,
FPRT_PR_4=0,
CPRL=0,
E1=0,
E2=0,
E3=0,
E4=0,
E5=0,
E6=0,
E7=0,
E8=0,
E9=0,
E10=0,
E11=0,
E12=0,
E13=0,
FMT_E1=0,
FMT_E2=0,
FMT_E3=0,
FMT_E4=0,
FMT_E5=0,
FMT_E6=0,
FMT_E7=0,
FMT_E8=0,
FMT_E9=0,
FMT_E10=0,
FMT_E11=0,
FMT_E12=0,
FMT_E13=0

Figure 76: Valeur initiale des variables du modèle.

A1.2.2 Séquenceur général

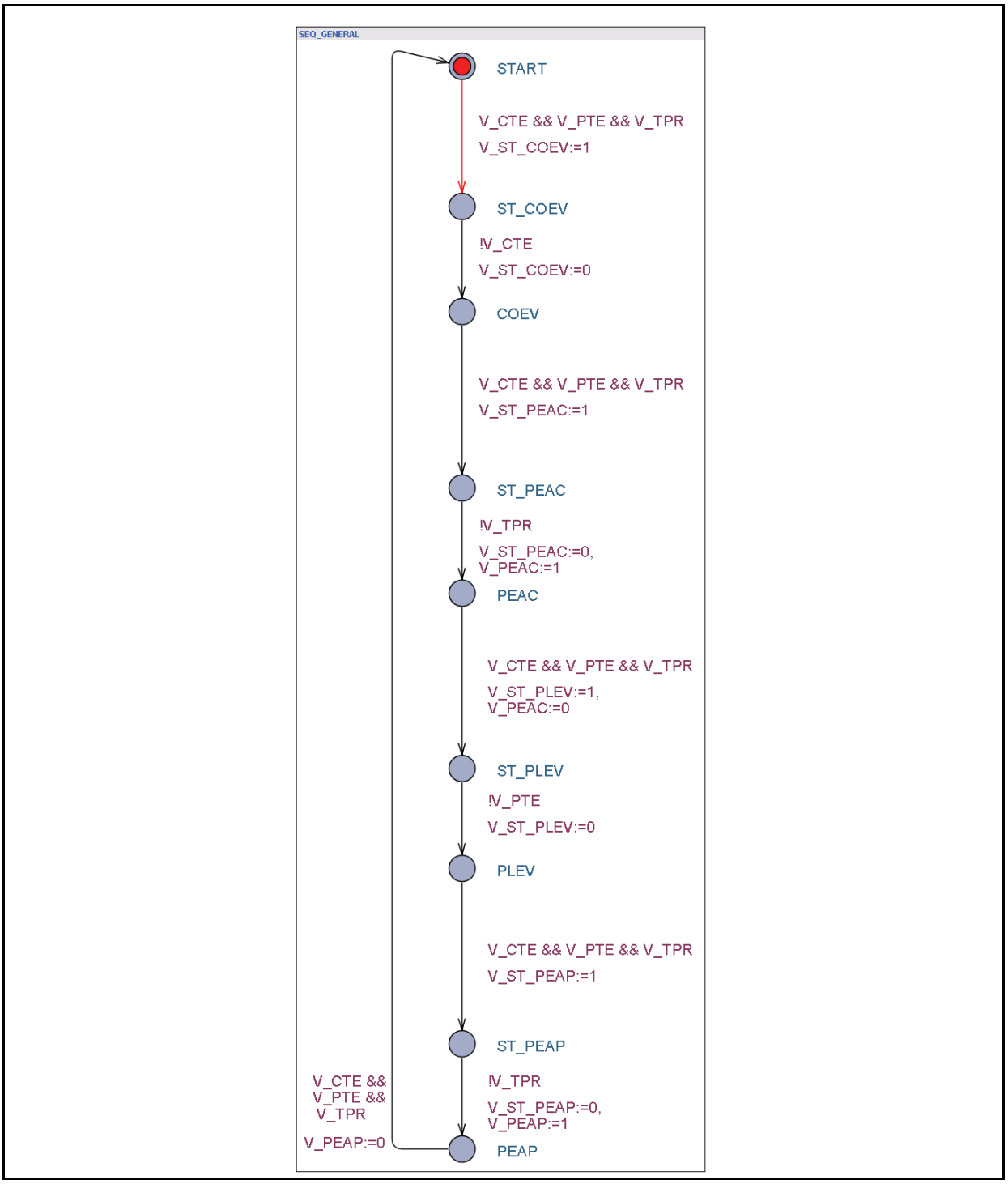


Figure 77: Séquenceur général (générique).

A1.2.3 Modèle du contrôleur

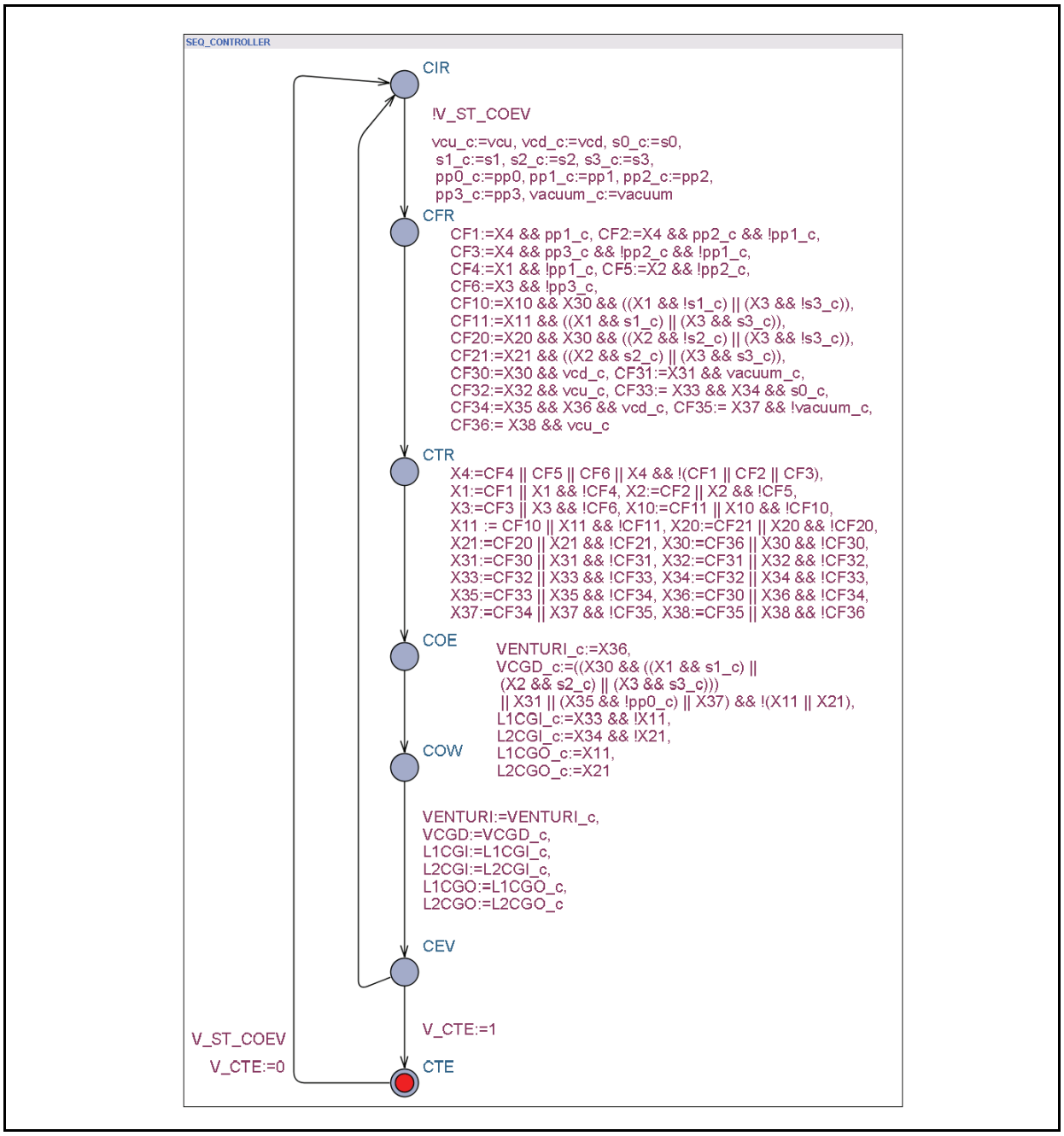


Figure 78: Séquenceur et modèle du contrôleur.

A1.2.4 Modèle du processus

A1.2.4.1 Séquenceur du processus

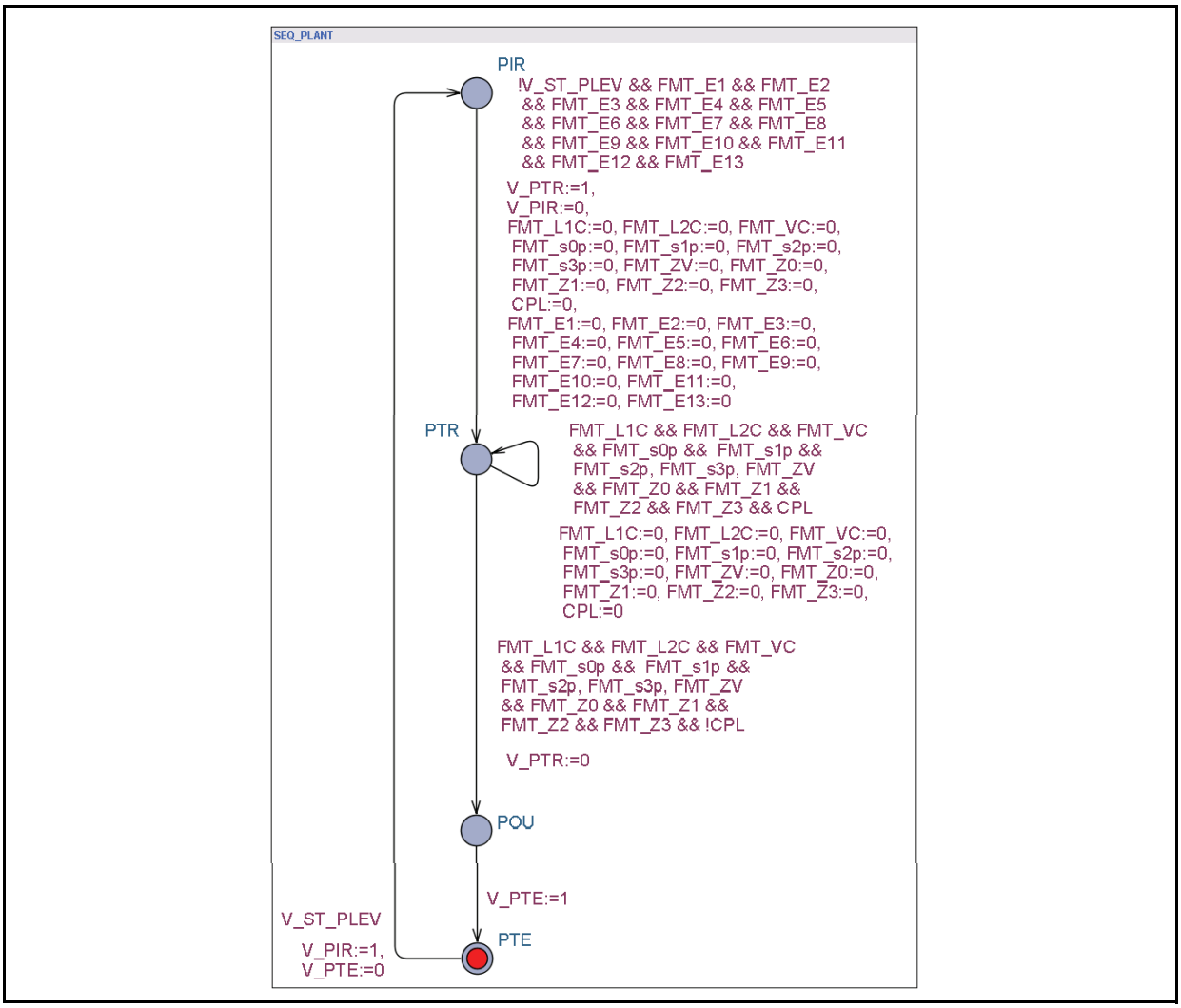


Figure 79: Séquenceur du processus

A1.2.4.2 Modules du processus

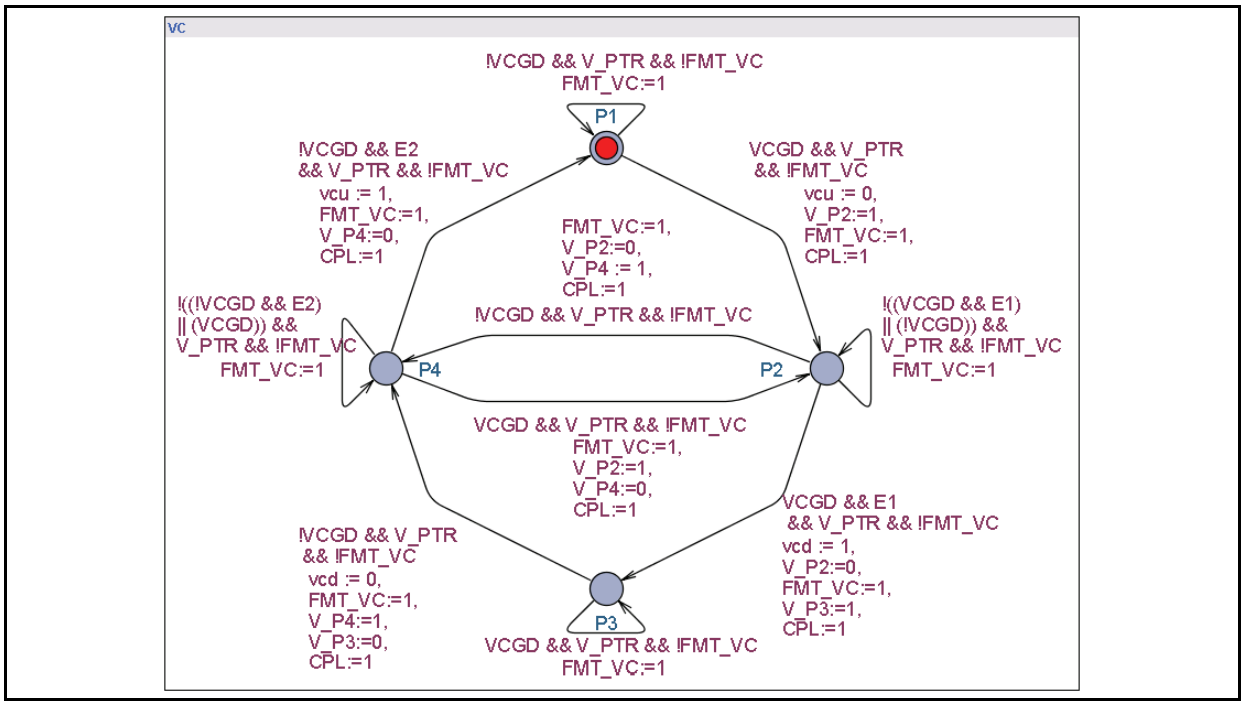


Figure 80: Modèle de la chaîne fonctionnelle correspondant au vérin vertical ; module «VC»

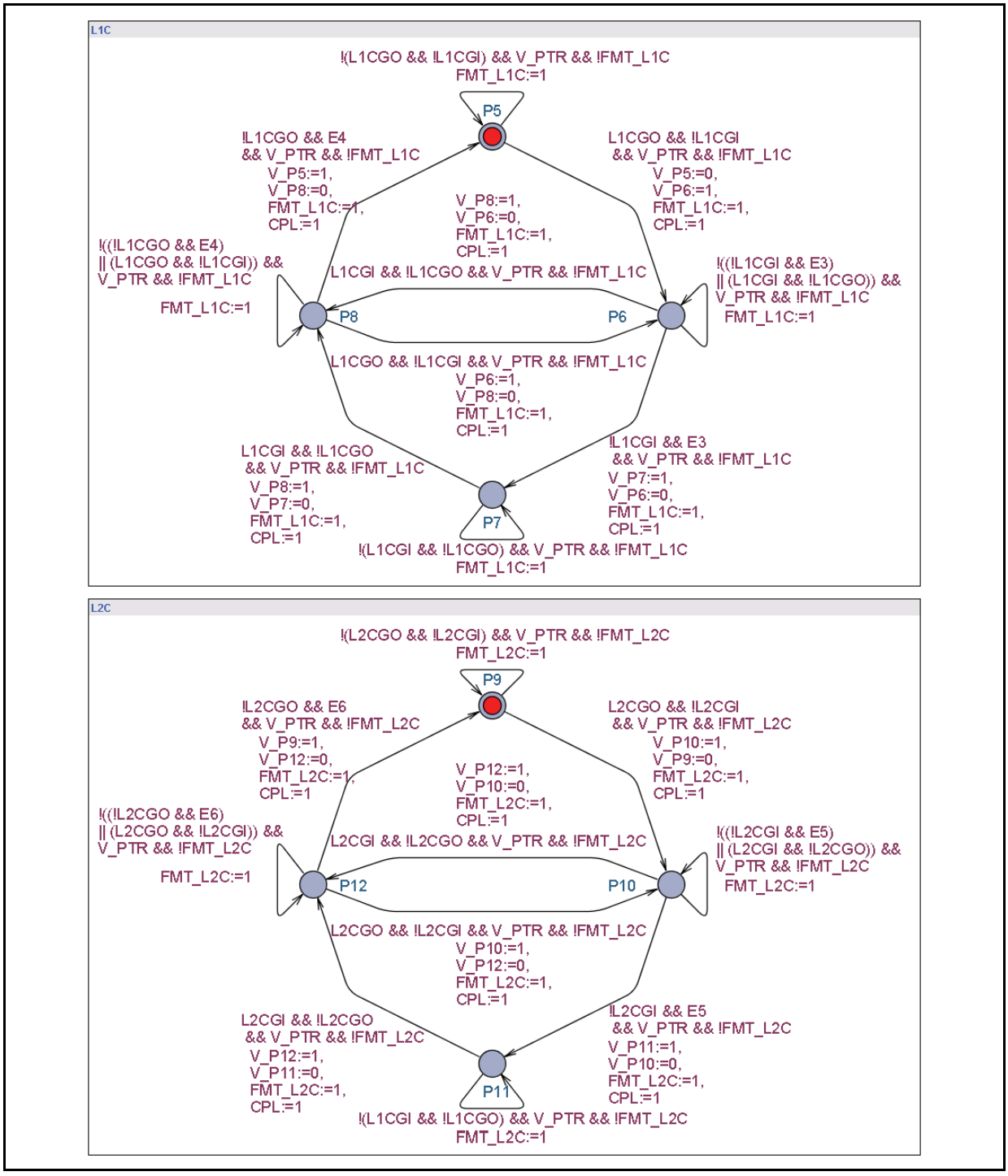


Figure 81: Modèles des chaînes fonctionnelles correspondant aux vérins horizontaux L1C et L2C; modules «L1C» et «L2C».

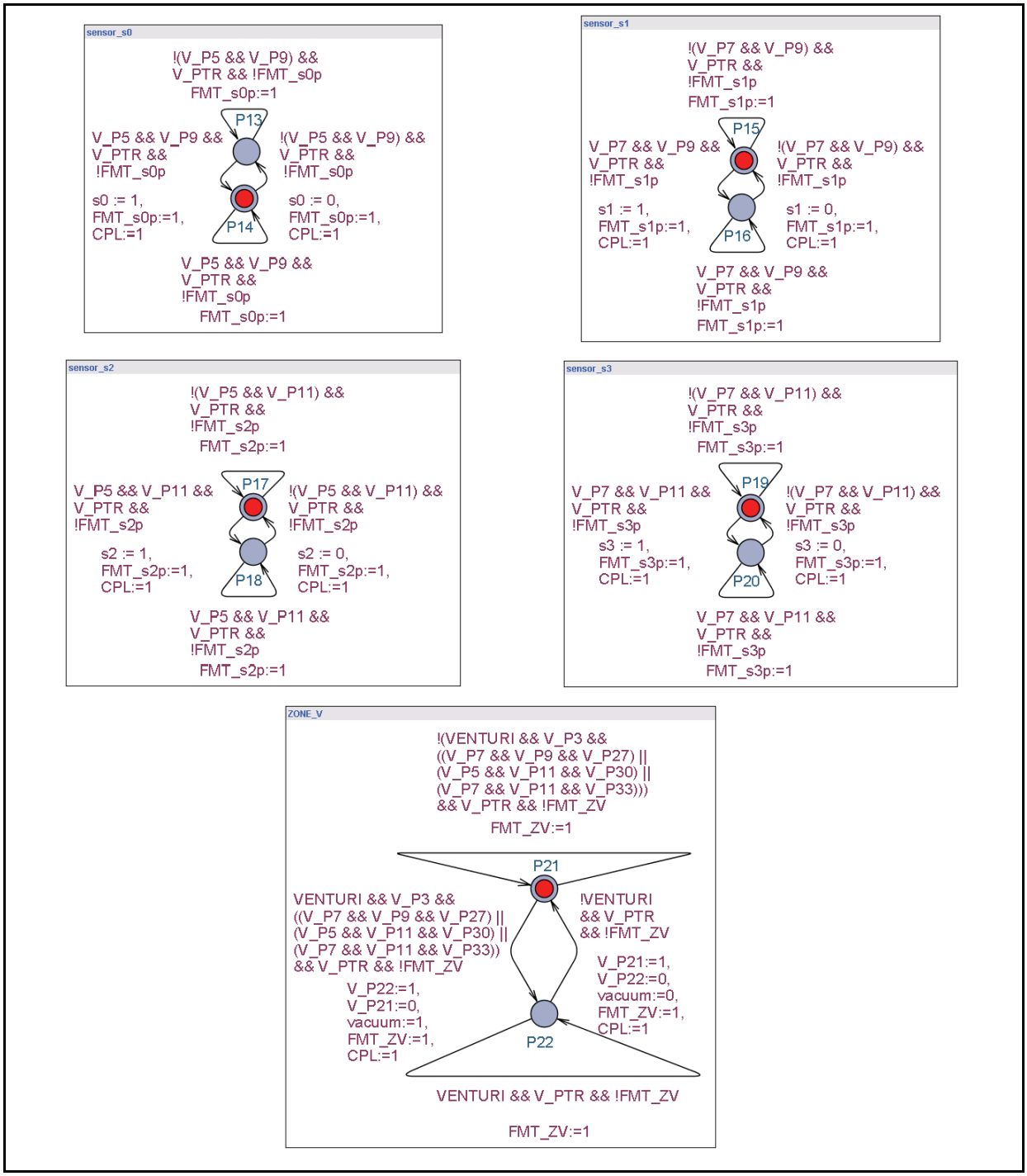


Figure 82: Modèles des capteurs s0, s1, s2, s3 et vacuum ;
modules «sensor_s0», «sensor_s1», «sensor_s2», «sensor_s3» et «ZONE_V».

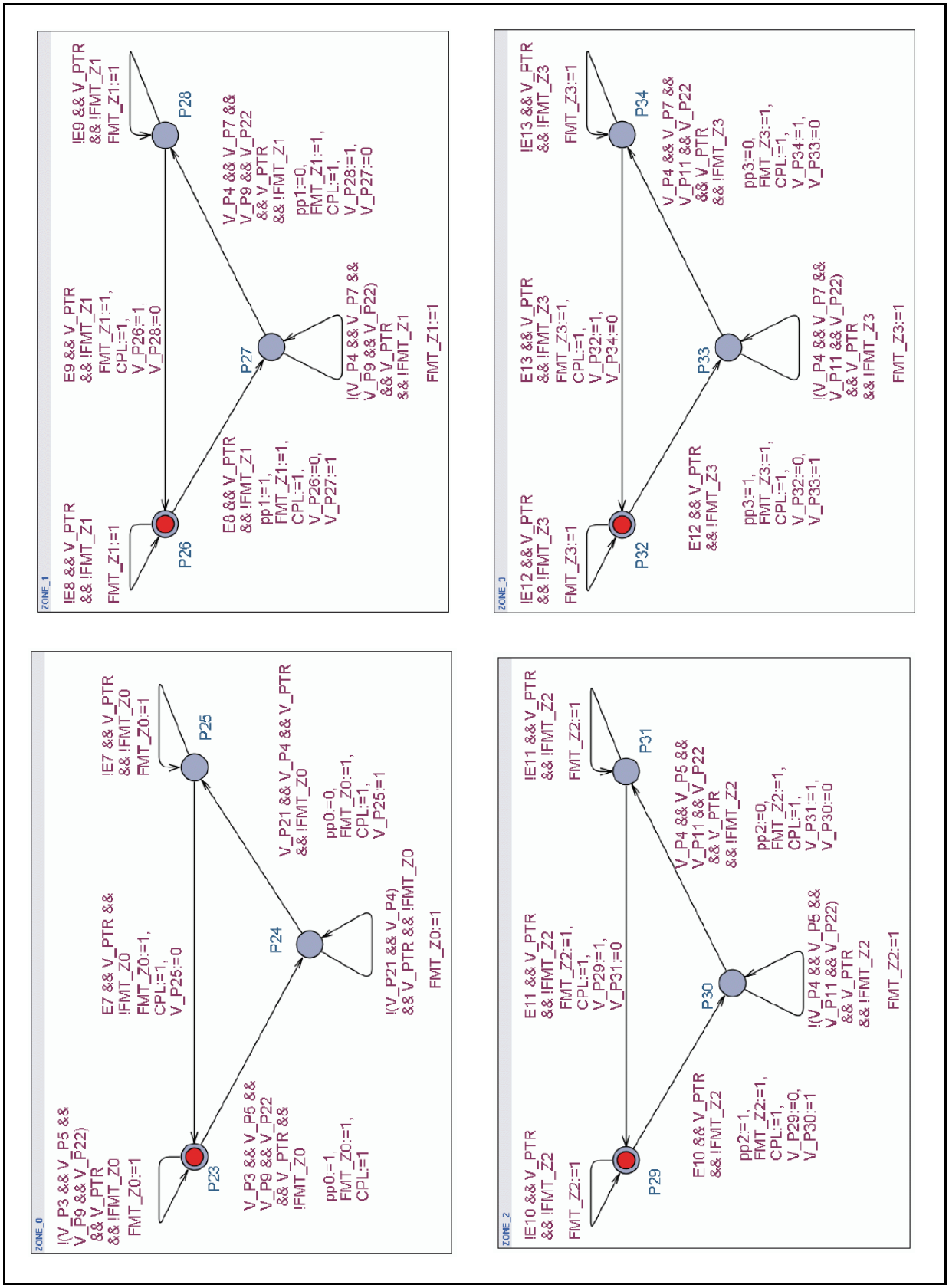


Figure 83: Modèles des capteurs pp0, pp1, pp2 et pp3 ;
modules «ZONE_0», «ZONE_1», «ZONE_2» et «ZONE_3».

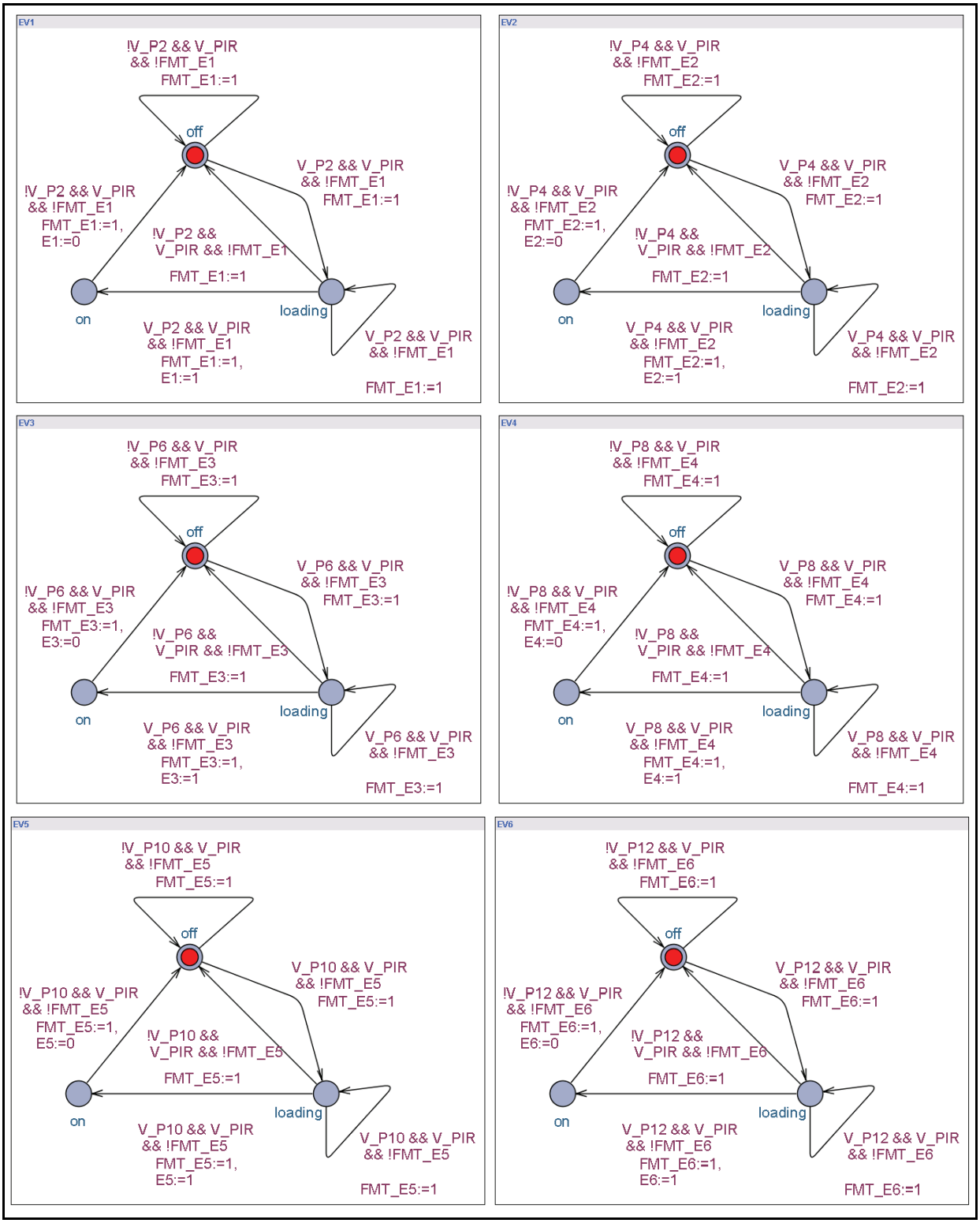


Figure 84: Modèles des temporisations logiques E1, E2, E3, E4, E5 et E6 ;
modules «EV1», «EV2», «EV3», «EV4», «EV5» et «EV6».

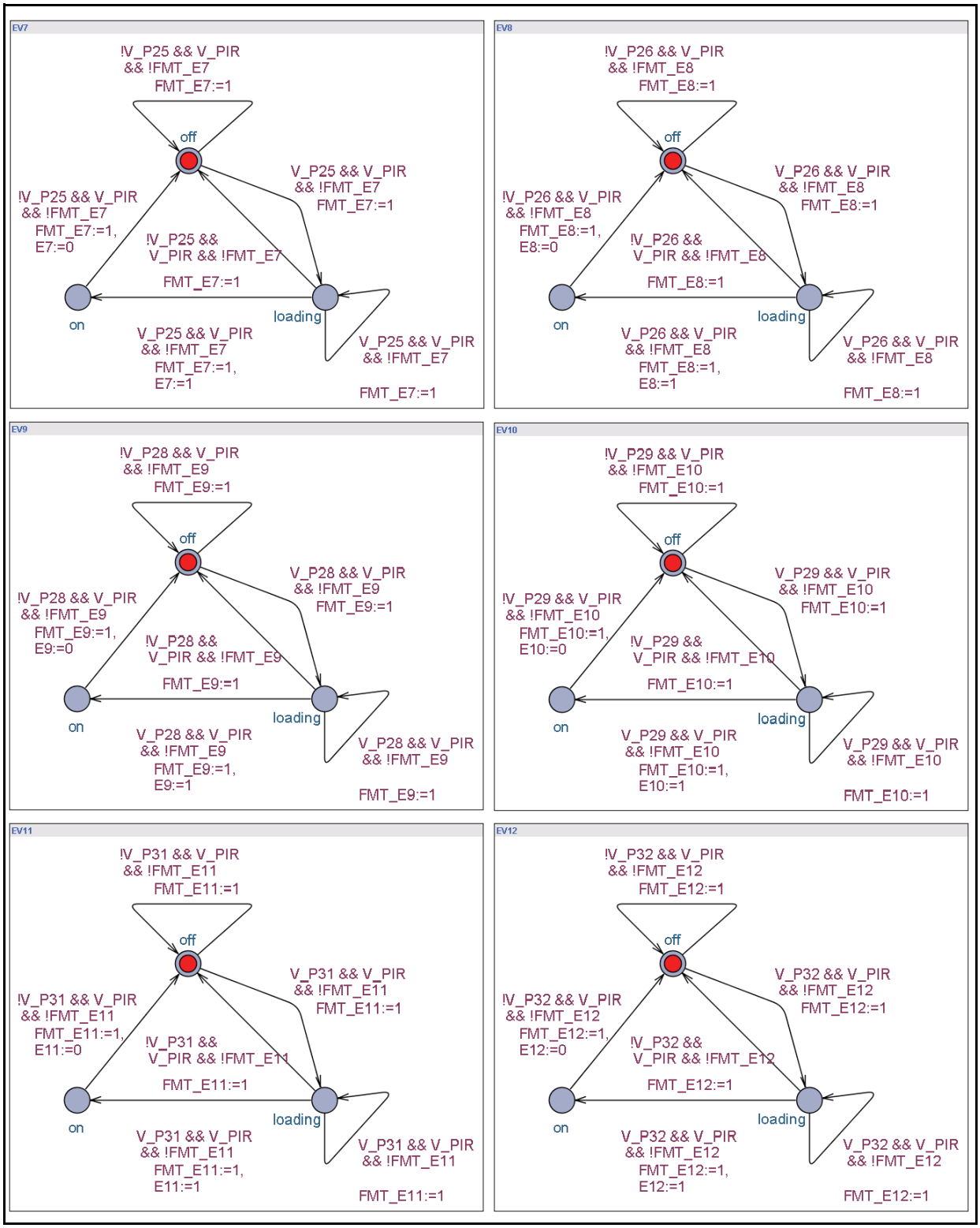


Figure 85: Modèles des temporisations logiques E7, E8, E9, E10, E11 et E12; modules «EV7», «EV8», «EV9», «EV10», «EV11» et «EV12».

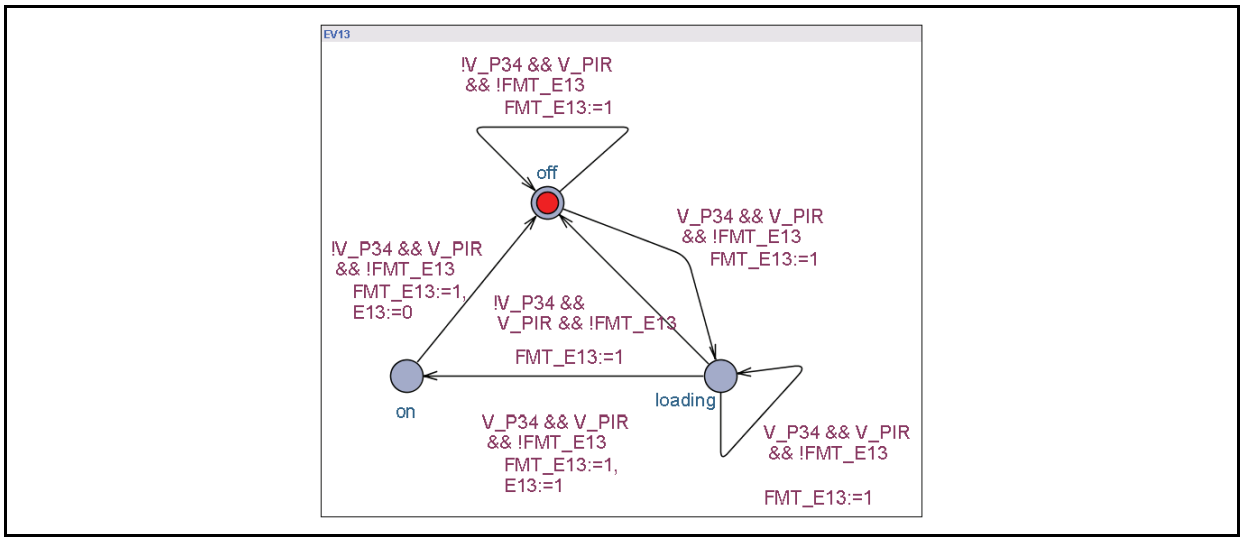


Figure 86: Modèle de la temporisation logique E13 ; module «EV13».

A1.2.5 Modèle des propriétés

A1.2.5.1 Formules en logique temporelle

L'expression formelle des propriétés à prouver doit d'une part être la traduction fidèle des propriétés exprimées en langage naturel et d'autre part intégrer les états du modèle où la preuve des propriétés doit être faite :

- après l'évolution du modèle du contrôleur (place PEAC du séquenceur général active, place CTE du séquenceur du contrôleur active, place PTE du séquenceur du processus active et place PTR du séquenceur des propriétés active), et
- après l'évolution du modèle du processus (place PEAP du séquenceur général active, place CTE du séquenceur du contrôleur active, place PTE du séquenceur du processus active et place PTR du séquenceur des propriétés active).

Ces états du modèle sont caractérisés par diverses variables d'état des séquenceurs qui nous permettent de définir les deux variables suivantes qui correspondent respectivement aux deux états stables requis pour les preuves :

$$PEAC = V\_CTE \wedge V\_CPE \wedge V\_TPR \wedge V\_PEAC$$

$$PEAP = V\_CTE \wedge V\_CPE \wedge V\_TPR \wedge V\_PEAP$$

La formalisation complète des propriétés est donnée tableau 7.

Tableau 7: Formalisation des propriétés aux états stables du modèles (PEAC et PEAP)

	Propriété en langage naturel avec sa formalisation en logique temporelle
PR_1.1	«Les ordres envoyés au distributeur qui commande le vérin L1C ne doivent jamais être émis en même temps»; $AG\neg(L1CGO \wedge L1CGI \wedge (PEAC \vee PEAP))$
PR_1.2	«Les ordres envoyés au distributeur qui commande le vérin L2C ne doivent jamais être émis en même temps» $AG\neg(L2CGO \wedge L2CGI \wedge (PEAC \vee PEAP))$

Tableau 7: Formalisation des propriétés aux états stables du modèles (PEAC et PEAP)

	Propriété en langage naturel avec sa formalisation en logique temporelle
PR_2	«Si l'ordre de descendre le vérin vertical est présent, alors on ne doit pas avoir l'émission d'ordres de commande relatifs aux mouvements des vérins horizontaux» $AG((VCGD \wedge (PEAC \vee PEAP)) \Rightarrow \neg(L1CGI \vee L1CGO \vee L2CGI \vee L2CGO))$
PR_3	«S'il y a émission d'un ordre de commande correspondant aux mouvements des vérins horizontaux alors le capteur «vcu» doit être à «1»» $AG(((L1CGI \vee L1CGO \vee L2CGI \vee L2CGO) \wedge (PEAC \vee PEAP)) \Rightarrow vcu)$
PR_5	«Il n'y a de mouvements horizontaux que si le capteur «vcu» est à «1»» $AG(((V\_P6 \vee V\_P6 \vee V\_P6 \vee V\_P6) \wedge (PEAC \vee PEAP)) \Rightarrow vcu)$
PR_6 : (PR_6.1 à PR_6.4, PR_6.10, PR_6.11, PR_6.20, PR_6.21, PR_6.30, à PR_6.38)	«Le modèle du contrôleur est toujours vivant» $AG(X1 \Rightarrow EF\neg X1)$ ... $AG(X4 \Rightarrow EF\neg X4)$ $AG(X10 \Rightarrow EF\neg X10)$ $AG(X11 \Rightarrow EF\neg X11)$ $AG(X20 \Rightarrow EF\neg X20)$ $AG(X21 \Rightarrow EF\neg X21)$ $AG(X30 \Rightarrow EF\neg X30)$ ... $AG(X38 \Rightarrow EF\neg X38)$
PR_7.1	«Si une pièce est détectée par pp1, alors on aura forcement, dans le futur, la sortie du vérin LC1» $AG((pp1 \wedge (PEAC \vee PEAP)) \Rightarrow EF(V\_P6 \wedge (PEAC \vee PEAP)))$
PR_7.2	«Si une pièce est détectée par pp2, alors on aura forcement, dans le futur, la sortie du vérin LC2» $AG((pp2 \wedge (PEAC \vee PEAP)) \Rightarrow EF(V\_P10 \wedge (PEAC \vee PEAP)))$
PR_7.3	«Si une pièce est détectée par pp3, alors on aura forcement, dans le futur, la sortie des vérins LC1 et LC2» $AG((pp3 \wedge (PEAC \vee PEAP)) \Rightarrow EF(V\_P6 \wedge V\_P10 \wedge (PEAC \vee PEAP)))$
PR_8.1	«Si une pièce est détectée par pp1, alors on aura forcement, dans le futur, l'aspiration de la pièce correspondante» $AG((pp1 \wedge (PEAC \vee PEAP)) \Rightarrow EF(s1 \wedge vcd \wedge vacuum \wedge pp1 \wedge (PEAC \vee PEAP)))$
PR_8.2	«Si une pièce est détectée par pp2, alors on aura forcement, dans le futur, l'aspiration de la pièce correspondante» $AG((pp2 \wedge (PEAC \vee PEAP)) \Rightarrow EF(s2 \wedge vcd \wedge vacuum \wedge pp1 \wedge (PEAC \vee PEAP)))$
PR_8.3	«Si une pièce est détectée par pp3, alors on aura forcement, dans le futur, l'aspiration de la pièce correspondante» $AG((pp3 \wedge (PEAC \vee PEAP)) \Rightarrow EF(s3 \wedge vcd \wedge vacuum \wedge pp1 \wedge (PEAC \vee PEAP)))$
PR_9	«Quand le vérin vertical descend, tous les autres vérins sont et restent en fin de course» $AG((V\_P2 \wedge (PEAC \vee PEAP)) \Rightarrow ((V\_P5 \wedge V\_P9) \vee (V\_P5 \wedge V\_P11) \vee (V\_P7 \wedge V\_P9) \vee (V\_P7 \wedge V\_P11)))$

A1.2.5.2 Séquenceur des propriétés

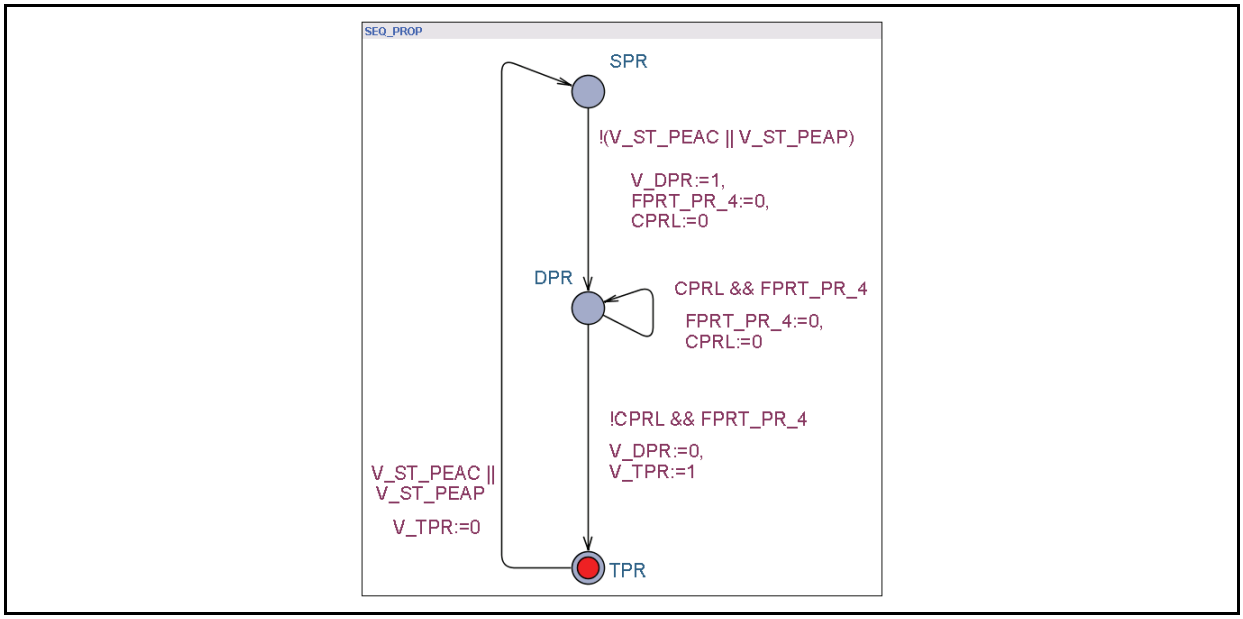


Figure 87: Séquenceur des propriétés; module «SEQ_PROP».

A1.2.5.3 Module des automates observateurs pour certaines propriétés.

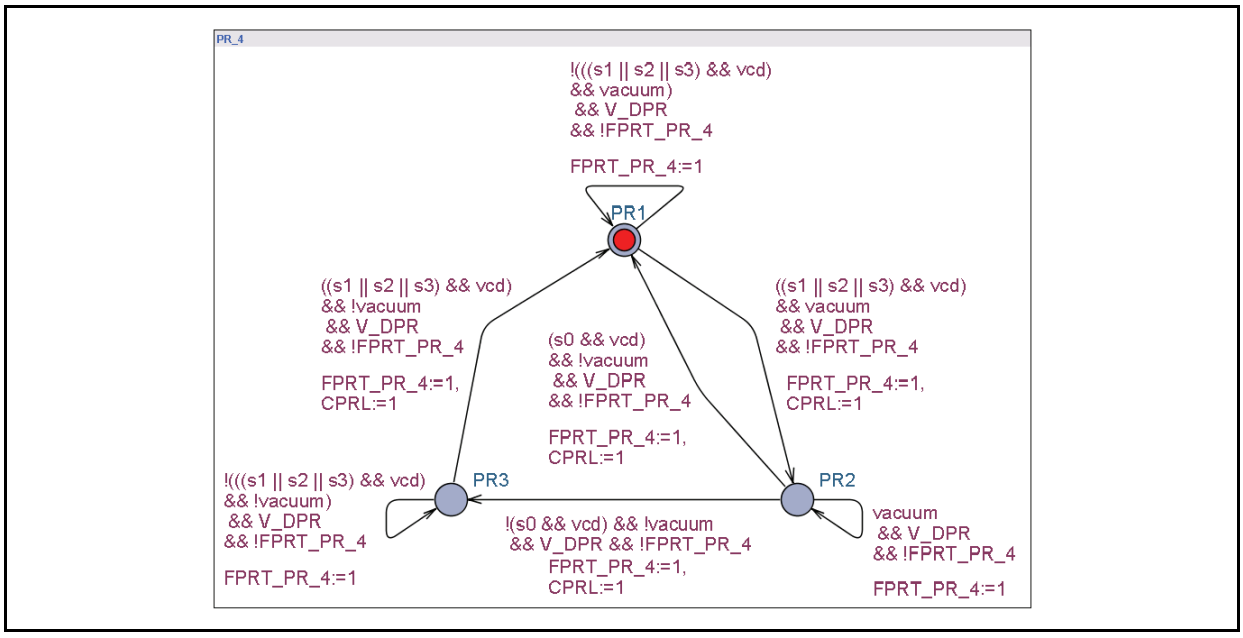


Figure 88: Automate observateur de la propriété PR_4 ; module «PR_4».

Annexe 2
Code NuSMV complet de l'exemple support utilisé

A2.1 Fichier «exemple_support-specif_OK_MB.nusmv»

```
MODULE main

IVAR
     FR : 1..26;

--**************************************
--******** SEQUENCEUR GENERAL ***********
--**************************************

--**************************************
--SEQUENCEUR GENERAL
VAR
     SEQ_GENERAL : {START, ST_COEV, COEV, ST_PEAC, PEAC, ST_PLEV, PLEV, ST_PEAP, PEAP};
DEFINE
     V_START := SEQ_GENERAL=START;
     V_ST_COEV := SEQ_GENERAL=ST_COEV;
     V_COEV := SEQ_GENERAL=COEV;
     V_ST_PEAC := SEQ_GENERAL=ST_PEAC;
     V_PEAC := SEQ_GENERAL=PEAC;
     V_ST_PLEV := SEQ_GENERAL=ST_PLEV;
     V_PLEV := SEQ_GENERAL=PLEV;
     V_ST_PEAP := SEQ_GENERAL=ST_PEAP;
     V_PEAP := SEQ_GENERAL=PEAP;

ASSIGN
     init(SEQ_GENERAL) := START;
     next(SEQ_GENERAL) := case
            SEQ_GENERAL=START & V_CTE & V_PTE & V_TPR : ST_COEV;
            SEQ_GENERAL=ST_COEV & !V_CTE : COEV;
            SEQ_GENERAL=COEV & V_CTE & V_PTE & V_TPR : ST_PEAC;
            SEQ_GENERAL=ST_PEAC & !V_TPR : PEAC;
            SEQ_GENERAL=PEAC & V_CTE & V_PTE & V_TPR : ST_PLEV;
            SEQ_GENERAL=ST_PLEV & !V_PTE : PLEV;
            SEQ_GENERAL=PLEV & V_CTE & V_PTE & V_TPR : ST_PEAP;
            SEQ_GENERAL=ST_PEAP & !V_TPR : PEAP;
            SEQ_GENERAL=PEAP & V_CTE & V_PTE & V_TPR : START;
            1 : SEQ_GENERAL;
            esac;

--**************************************
--SEQUENCEUR DU PROCESSUS

VAR
     SEQ_PLANT : {PIR, PTR, POU, PTE};
DEFINE
     V_PIR := SEQ_PLANT=PIR;
     V_PTR := SEQ_PLANT=PTR;
     V_POU := SEQ_PLANT=POU;
     V_PTE := SEQ_PLANT=PTE;
ASSIGN
     init(SEQ_PLANT) := PTE;
     next(SEQ_PLANT) := case
            SEQ_PLANT=PTE & V_ST_PLEV : PIR;

            SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13 : PTR;

            SEQ_PLANT=PTR & CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                   & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3 : PTR;

            SEQ_PLANT=PTR & !CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                   & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3 : POU;

            SEQ_PLANT=POU : PTE;
            1 : SEQ_PLANT;
            esac;

--**************************************
--SEQUENCEUR DES PROPRIÉTÉS
VAR
     SEQ_PROP : {SPR, DPR, TPR};
DEFINE
     V_SPR := SEQ_PROP=SPR;
     V_DPR := SEQ_PROP=DPR;
     V_TPR := SEQ_PROP=TPR;
ASSIGN
     init(SEQ_PROP) := TPR;
     next(SEQ_PROP) := case
            SEQ_PROP=TPR & (V_ST_PEAC | V_ST_PEAP) : SPR;
            SEQ_PROP=SPR & !(V_ST_PEAC | V_ST_PEAP) : DPR;
            SEQ_PROP=DPR & !CPRL & FPRT_PR_4 : TPR;
            SEQ_PROP=DPR & CPRL & FPRT_PR_4 : DPR;
            1 : SEQ_PROP;
            esac;
```

```
--**************************************
--SEQUENCEUR DU CONTRÔLEUR : STRUCTURE

VAR
     SEQ_CONTROLLER : {CIR, CFR, CTR, COE, COW, CEV, CTE};
DEFINE
     V_CIR := SEQ_CONTROLLER=CIR;
     V_CFR := SEQ_CONTROLLER=CFR;
     V_CTR := SEQ_CONTROLLER=CTR;
     V_COE := SEQ_CONTROLLER=COE;
     V_COW := SEQ_CONTROLLER=COW;
     V_CEV := SEQ_CONTROLLER=CEV;
     V_CTE := SEQ_CONTROLLER=CTE;
ASSIGN
     init(SEQ_CONTROLLER) := CTE;
     next(SEQ_CONTROLLER) := case
            SEQ_CONTROLLER=CTE & V_ST_COEV : CIR;
            SEQ_CONTROLLER=CIR & !V_ST_COEV : CFR;
            SEQ_CONTROLLER=CFR : CTR;
            SEQ_CONTROLLER=CTR : COE;
            SEQ_CONTROLLER=COE : COW;
            SEQ_CONTROLLER=COW : CEV;
            SEQ_CONTROLLER=CEV & FR=1 : CIR;
            SEQ_CONTROLLER=CEV & FR=2 : CTE;
            1 : SEQ_CONTROLLER;
            esac;

--**************************************************
-- SEQUENCEUR DU CONTRÔLEUR : TRADUCTION DE LA SPECIFICATION

VAR
     vcu : boolean;
     vcd : boolean;
     s0 : boolean;
     s1 : boolean;
     s2 : boolean;
     s3 : boolean;
     pp0 : boolean;
     pp1 : boolean;
     pp2 : boolean;
     pp3 : boolean;
     vacuum : boolean;

     vcu_c : boolean;
     vcd_c : boolean;
     s0_c : boolean;
     s1_c : boolean;
     s2_c : boolean;
     s3_c : boolean;
     pp0_c : boolean;
     pp1_c : boolean;
     pp2_c : boolean;
     pp3_c : boolean;
     vacuum_c : boolean;

     CF1 : boolean;
     CF2 : boolean;
     CF3 : boolean;
     CF4 : boolean;
     CF5 : boolean;
     CF6 : boolean;

     CF10 : boolean;
     CF11 : boolean;

     CF20 : boolean;
     CF21 : boolean;

     CF30 : boolean;
     CF31 : boolean;
     CF32 : boolean;
     CF33 : boolean;
     CF34 : boolean;
     CF35 : boolean;
     CF36 : boolean;

     X1 : boolean;
     X2 : boolean;
     X3 : boolean;
     X4 : boolean;

     X10 : boolean;
     X11 : boolean;

     X20 : boolean;
     X21 : boolean;

     X30 : boolean;
```

```
      X31 : boolean;
      X32 : boolean;
      X33 : boolean;
      X34 : boolean;
      X35 : boolean;
      X36 : boolean;
      X37 : boolean;
      X38 : boolean;

      VCGD : boolean;
      L1CGO : boolean;
      L1CGI : boolean;
      L2CGO : boolean;
      L2CGI : boolean;
      VENTURI : boolean;

      VCGD_c : boolean;
      L1CGO_c : boolean;
      L1CGI_c : boolean;
      L2CGO_c : boolean;
      L2CGI_c : boolean;
      VENTURI_c : boolean;

ASSIGN
      init(vcu_c) := 0;
      next(vcu_c) := case SEQ_CONTROLLER=CIR & !V_ST_COEV : vcu; 1 : vcu_c; esac;

      init(vcd_c) := 0;
      next(vcd_c) := case SEQ_CONTROLLER=CIR & !V_ST_COEV : vcd; 1 : vcd_c; esac;

      init(s0_c) := 0;
      next(s0_c) := case SEQ_CONTROLLER=CIR & !V_ST_COEV : s0; 1 : s0_c; esac;

      init(s1_c) := 0;
      next(s1_c) := case SEQ_CONTROLLER=CIR & !V_ST_COEV : s1; 1 : s1_c; esac;

      init(s2_c) := 0;
      next(s2_c) := case SEQ_CONTROLLER=CIR & !V_ST_COEV : s2; 1 : s2_c; esac;

      init(s3_c) := 0;
      next(s3_c) := case SEQ_CONTROLLER=CIR & !V_ST_COEV : s3; 1 : s3_c; esac;

      init(pp0_c) := 0;
      next(pp0_c) := case SEQ_CONTROLLER=CIR & !V_ST_COEV : pp0; 1 : pp0_c; esac;

      init(pp1_c) := 0;
      next(pp1_c) := case SEQ_CONTROLLER=CIR & !V_ST_COEV : pp1; 1 : pp1_c; esac;

      init(pp2_c) := 0;
      next(pp2_c) := case SEQ_CONTROLLER=CIR & !V_ST_COEV : pp2; 1 : pp2_c; esac;

      init(pp3_c) := 0;
      next(pp3_c) := case SEQ_CONTROLLER=CIR & !V_ST_COEV : pp3; 1 : pp3_c; esac;

      init(vacuum_c) := 0;
      next(vacuum_c) := case SEQ_CONTROLLER=CIR & !V_ST_COEV : vacuum; 1 : vacuum_c; esac;

ASSIGN
      init(CF1) := 0;
      next(CF1) := case SEQ_CONTROLLER=CFR : X4 & pp1_c; 1 : CF1; esac;

      init(CF2) := 0;
      next(CF2) := case SEQ_CONTROLLER=CFR : X4 & pp2_c & !pp1_c; 1 : CF2; esac;

      init(CF3) := 0;
      next(CF3) := case SEQ_CONTROLLER=CFR : X4 & pp3_c & !pp2_c & !pp1_c; 1 : CF3; esac;

      init(CF4) := 0;
      next(CF4) := case SEQ_CONTROLLER=CFR : X1 & !pp1_c; 1 : CF4; esac;

      init(CF5) := 0;
      next(CF5) := case SEQ_CONTROLLER=CFR : X2 & !pp2_c; 1 : CF5; esac;

      init(CF6) := 0;
      next(CF6) := case SEQ_CONTROLLER=CFR : X3 & !pp3_c; 1 : CF6; esac;

      init(CF10) := 0;
      next(CF10) := case SEQ_CONTROLLER=CFR : X10 & X30 & ((X1 & !s1_c) | (X3 & !s3_c)); 1 : CF10; esac;

      init(CF11) := 0;
      next(CF11) := case SEQ_CONTROLLER=CFR : X11 & ((X1 & s1_c) | (X3 & s3_c)); 1 : CF11; esac;

      init(CF20) := 0;
      next(CF20) := case SEQ_CONTROLLER=CFR : X20 & X30 & ((X2 & !s2_c) | (X3 & !s3_c)); 1 : CF20; esac;

      init(CF21) := 0;
      next(CF21) := case SEQ_CONTROLLER=CFR : X21 & ((X2 & s2_c) | (X3 & s3_c)); 1 : CF21; esac;

      init(CF30) := 0;
```

```
      next(CF30) := case SEQ_CONTROLLER=CFR : X30 & vcd_c; 1 : CF30; esac;

      init(CF31) := 0;
      next(CF31) := case SEQ_CONTROLLER=CFR : X31 & vacuum_c; 1 : CF31; esac;

      init(CF32) := 0;
      next(CF32) := case SEQ_CONTROLLER=CFR : X32 & vcu_c; 1 : CF32; esac;

      init(CF33) := 0;
      next(CF33) := case SEQ_CONTROLLER=CFR : X33 & X34 & s0_c; 1 : CF33; esac;

      init(CF34) := 0;
      next(CF34) := case SEQ_CONTROLLER=CFR : X35 & X36 & vcd_c; 1 : CF34; esac;

      init(CF35) := 0;
      next(CF35) := case SEQ_CONTROLLER=CFR : X37 & !vacuum_c; 1 : CF35; esac;

      init(CF36) := 0;
      next(CF36) := case SEQ_CONTROLLER=CFR : X38 & vcu_c; 1 : CF36; esac;

ASSIGN
      init(X1) := 0;
      next(X1) := case SEQ_CONTROLLER=CTR : CF1 | (X1 & !CF4); 1 : X1; esac;

      init(X2) := 0;
      next(X2) := case SEQ_CONTROLLER=CTR : CF2 | (X2 & !CF5); 1 : X2; esac;

      init(X3) := 0;
      next(X3) := case SEQ_CONTROLLER=CTR : CF3 | (X3 & !CF6); 1 : X3; esac;

      init(X4) := 1;
      next(X4) := case SEQ_CONTROLLER=CTR : CF4 | CF5 | CF6 | (X4 & !(CF1 | CF2 | CF3)); 1 : X4; esac;

      init(X10) := 1;
      next(X10) := case SEQ_CONTROLLER=CTR : CF11 | (X10 & !CF10); 1 : X10; esac;

      init(X11) := 0;
      next(X11) := case SEQ_CONTROLLER=CTR : CF10 | (X11 & !CF11); 1 : X11; esac;

      init(X20) := 1;
      next(X20) := case SEQ_CONTROLLER=CTR : CF21 | (X20 & !CF20); 1 : X20; esac;

      init(X21) := 0;
      next(X21) := case SEQ_CONTROLLER=CTR : CF20 | (X21 & !CF21); 1 : X21; esac;

      init(X30) := 1;
      next(X30) := case SEQ_CONTROLLER=CTR : CF36 | (X30 & !CF30); 1 : X30; esac;

      init(X31) := 0;
      next(X31) := case SEQ_CONTROLLER=CTR : CF30 | (X31 & !CF31); 1 : X31; esac;

      init(X32) := 0;
      next(X32) := case SEQ_CONTROLLER=CTR : CF31 | (X32 & !CF32); 1 : X32; esac;

      init(X33) := 0;
      next(X33) := case SEQ_CONTROLLER=CTR : CF32 | (X33 & !CF33); 1 : X33; esac;

      init(X34) := 0;
      next(X34) := case SEQ_CONTROLLER=CTR : CF32 | (X34 & !CF33); 1 : X34; esac;

      init(X35) := 0;
      next(X35) := case SEQ_CONTROLLER=CTR : CF33 | (X35 & !CF34); 1 : X35; esac;

      init(X36) := 0;
      next(X36) := case SEQ_CONTROLLER=CTR : CF30 | (X36 & !CF34); 1 : X36; esac;

      init(X37) := 0;
      next(X37) := case SEQ_CONTROLLER=CTR : CF34 | (X37 & !CF35); 1 : X37; esac;

      init(X38) := 0;
      next(X38) := case SEQ_CONTROLLER=CTR : CF35 | (X38 & !CF36); 1 : X38; esac;

ASSIGN
      init(L2CGO_c) := 0;
      next(L2CGO_c) := case SEQ_CONTROLLER=COE : X21; 1 : L2CGO_c; esac;

      init(L2CGI_c) := 0;
      next(L2CGI_c) := case SEQ_CONTROLLER=COE : X34 & !X21; 1 : L2CGI_c; esac;

      init(L1CGO_c) := 0;
      next(L1CGO_c) := case SEQ_CONTROLLER=COE : X11; 1 : L1CGO_c; esac;

      init(L1CGI_c) := 0;
      next(L1CGI_c) := case SEQ_CONTROLLER=COE : X33 & !X11; 1 : L1CGI_c; esac;

      init(VENTURI_c) := 0;
      next(VENTURI_c) := case SEQ_CONTROLLER=COE : X36; 1 : VENTURI_c; esac;

      init(VCGD_c) := 0;
```

```
     next(VCGD_c) := case SEQ_CONTROLLER=COE : ((X30 & ((X1 & s1_c) | (X2 & s2_c) | (X3 & s3_c)))
                         | X31 | (X35 & !pp0_c) | X37) & !(X11 | X21)); 1 : VCGD_c; esac;

ASSIGN

     init(L2CGO) := 0;
     next(L2CGO) := case SEQ_CONTROLLER=COW : L2CGO_c; 1 : L2CGO; esac;

     init(L2CGI) := 0;
     next(L2CGI) := case SEQ_CONTROLLER=COW : L2CGI_c; 1 : L2CGI; esac;

     init(L1CGO) := 0;
     next(L1CGO) := case SEQ_CONTROLLER=COW : L1CGO_c; 1 : L1CGO; esac;

     init(L1CGI) := 0;
     next(L1CGI) := case SEQ_CONTROLLER=COW : L1CGI_c; 1 : L1CGI; esac;

     init(VENTURI) := 0;
     next(VENTURI) := case SEQ_CONTROLLER=COW : VENTURI_c; 1 : VENTURI; esac;

     init(VCGD) := 0;
     next(VCGD) := case SEQ_CONTROLLER=COW : VCGD_c; 1 : VCGD; esac;

--*****************************************************
--******************* PROPRIÉTÉS **********************
--*****************************************************

--**************************************
--PR_1.1 Les ordres envoyés au distributeur qui commande le vérin L1C ne doivent jamais être émis en même temps

SPEC
     AG !(L1CGI & L1CGO & V_CTE & V_PTE & V_TPR & V_PEAC)
SPEC
     AG !(L1CGI & L1CGO & V_CTE & V_PTE & V_TPR & V_PEAP)

--PR_1.2 Les ordres envoyés au distributeur qui commande le vérin L2C ne doivent jamais être émis en même temps

SPEC
     AG !(L2CGI & L2CGO & V_CTE & V_PTE & V_TPR & V_PEAC)
SPEC
     AG !(L2CGI & L2CGO & V_CTE & V_PTE & V_TPR & V_PEAP)

--**************************************
--PR_2 Si l'ordre de descendre le vérin vertical est présent, alors on ne doit pas avoir l'émission d'ordres de
     commande relatifs aux mouvements des vérins horizontaux

SPEC
     AG ((VCGD & V_CTE & V_PTE & V_TPR & V_PEAC) -> !(L1CGI | L1CGO | L2CGI | L2CGO))

SPEC
     AG ((VCGD & V_CTE & V_PTE & V_TPR & V_PEAP) -> !(L1CGI | L1CGO | L2CGI | L2CGO))

--**************************************
--PR_3 S'il y a émission d'un ordre de commande correspondant aux mouvements des vérins horizontaux alors le capteur
     «vcu» doit être à «1»

SPEC
     AG(((L1CGI | L1CGO | L2CGI | L2CGO) & V_CTE & V_PTE & V_TPR & V_PEAC) -> vcu)

SPEC
     AG(((L1CGI | L1CGO | L2CGI | L2CGO) & V_CTE & V_PTE & V_TPR & V_PEAP) -> vcu)

--**************************************
--PR_4 Après avoir été saisie, une pièce ne peut être lâchée qu'au droit de sa position de dépose

VAR
     PR_4 : {PR1, PR2, PR3};
     FPRT_PR_4 : boolean;

DEFINE
     V_PR1 := PR_4=PR1;
     V_PR2 := PR_4=PR2;
     V_PR3 := PR_4=PR3;

ASSIGN
     init(PR_4) := PR1;
     next(PR_4) := case
           PR_4=PR1 & !(((s1 | s2 | s3) & vcd) & vacuum) & V_DPR & !FPRT_PR_4 : PR1;
           PR_4=PR1 & ((s1 | s2 | s3) & vcd) & vacuum & V_DPR & !FPRT_PR_4 : PR2;
           PR_4=PR2 & (s0 & vcd) & !vacuum & V_DPR & !FPRT_PR_4 : PR1;
           PR_4=PR2 & vacuum & V_DPR & !FPRT_PR_4 : PR2;
           PR_4=PR2 & !(s0 & vcd) & !vacuum & V_DPR & !FPRT_PR_4 : PR3;
           PR_4=PR3 & ((s1 | s2 | s3) & vcd) & !vacuum & V_DPR & !FPRT_PR_4 : PR1;
           PR_4=PR3 & !(((s1 | s2 | s3) & vcd) & !vacuum) & V_DPR & !FPRT_PR_4 : PR3;
           1 : PR_4;
           esac;

     init(FPRT_PR_4) := 0;
     next(FPRT_PR_4) := case
```

```
            (PR_4=PR1 & !(((s1 | s2 | s3) & vcd) & vacuum) & V_DPR & !FPRT_PR_4) |
            (PR_4=PR1 & ((s1 | s2 | s3) & vcd) & vacuum & V_DPR & !FPRT_PR_4) |
            (PR_4=PR2 & (s0 & vcd) & !vacuum & V_DPR & !FPRT_PR_4) |
            (PR_4=PR2 & vacuum & V_DPR & !FPRT_PR_4) |
            (PR_4=PR2 & !(s0 & vcd) & !vacuum & V_DPR & !FPRT_PR_4) |
            (PR_4=PR3 & ((s1 | s2 | s3) & vcd) & !vacuum & V_DPR & !FPRT_PR_4) |
            (PR_4=PR3 & !(((s1 | s2 | s3) & vcd) & !vacuum) & V_DPR & !FPRT_PR_4) : 1;
            (SEQ_PROP=SPR & !(V_ST_PEAC | V_ST_PEAP)) |
            (SEQ_PROP=DPR & CPRL & FPRT_PR_4) : 0;
            1 : FPRT_PR_4;
            esac;

SPEC
     AG !V_PR3

--*****************************************
--PR_5 Il n'y a de mouvements horizontaux que si le capteur «vcu» est à «1»

SPEC
     AG(((V_P6 | V_P8 | V_P10 | V_P12) & V_CTE & V_PTE & V_TPR & V_PEAC) -> vcu)

SPEC
     AG(((V_P6 | V_P8 | V_P10 | V_P12) & V_CTE & V_PTE & V_TPR & V_PEAP) -> vcu)

--*****************************************
--PR_6 Le modèle du contrôleur est toujours vivant

SPEC
     AG (X1 -> EF !X1)
SPEC
     AG (X2 -> EF !X2)
SPEC
     AG (X3 -> EF !X3)
SPEC
     AG (X4 -> EF !X4)

SPEC
     AG (X10 -> EF !X10)
SPEC
     AG (X11 -> EF !X11)

SPEC
     AG (X20 -> EF !X20)
SPEC
     AG (X21 -> EF !X21)

SPEC
     AG (X30 -> EF !X30)
SPEC
     AG (X31 -> EF !X31)
SPEC
     AG (X32 -> EF !X32)
SPEC
     AG (X33 -> EF !X33)
SPEC
     AG (X34 -> EF !X34)
SPEC
     AG (X35 -> EF !X35)
SPEC
     AG (X36 -> EF !X36)
SPEC
     AG (X37 -> EF !X37)
SPEC
     AG (X38 -> EF !X38)

--*****************************************
--PR_7.1 Si une pièce est détectée par pp1, alors on aura forcement, dans le futur, la sortie du vérin LC1

SPEC
     AG ((pp1 & V_CTE & V_PTE & V_TPR & V_PEAC) -> EF ((V_P6) & V_CTE & V_PTE & V_TPR & V_PEAC))

SPEC
     AG ((pp1 & V_CTE & V_PTE & V_TPR & V_PEAP) -> EF ((V_P6) & V_CTE & V_PTE & V_TPR & V_PEAP))

--PR_7.2 Si une pièce est détectée par pp2, alors on aura forcement, dans le futur, la sortie du vérin LC2

SPEC
     AG ((pp2 & V_CTE & V_PTE & V_TPR & V_PEAC) -> EF ((V_P10) & V_CTE & V_PTE & V_TPR & V_PEAC))

SPEC
     AG ((pp2 & V_CTE & V_PTE & V_TPR & V_PEAP) -> EF ((V_P10) & V_CTE & V_PTE & V_TPR & V_PEAP))

--PR_7.3 Si une pièce est détectée par pp3, alors on aura forcement, dans le futur, la sortie des vérins LC1 et LC2

SPEC
     AG ((pp3 & V_CTE & V_PTE & V_TPR & V_PEAC) -> EF ((V_P6 & V_P10) & V_CTE & V_PTE & V_TPR & V_PEAC))

SPEC
     AG ((pp3 & V_CTE & V_PTE & V_TPR & V_PEAP) -> EF ((V_P6 & V_P10) & V_CTE & V_PTE & V_TPR & V_PEAP))
```

```
--*****************************************
--PR_8.1 Si une pièce est détectée par pp1, alors on aura forcement, dans le futur, l'aspiration de la pièce
     correspondante

SPEC
     AG ((pp1 & V_CTE & V_PTE & V_TPR & V_PEAC) -> EF ((s1 & vcd & vacuum) & V_CTE & V_PTE & V_TPR & V_PEAC))

SPEC
     AG ((pp1 & V_CTE & V_PTE & V_TPR & V_PEAP) -> EF ((s1 & vcd & vacuum) & V_CTE & V_PTE & V_TPR & V_PEAP))

--PR_8.2 Si une pièce est détectée par pp2, alors on aura forcement, dans le futur, l'aspiration de la pièce
     correspondante

SPEC
     AG ((pp2 & V_CTE & V_PTE & V_TPR & V_PEAC) -> EF ((s2 & vcd & vacuum) & V_CTE & V_PTE & V_TPR & V_PEAC))

SPEC
     AG ((pp2 & V_CTE & V_PTE & V_TPR & V_PEAP) -> EF ((s2 & vcd & vacuum) & V_CTE & V_PTE & V_TPR & V_PEAP))

--PR_8.3 Si une pièce est détectée par pp3, alors on aura forcement, dans le futur, l'aspiration de la pièce
     correspondante

SPEC
     AG ((pp3 & V_CTE & V_PTE & V_TPR & V_PEAC) -> EF ((s3 & vcd & vacuum) & V_CTE & V_PTE & V_TPR & V_PEAC))

SPEC
     AG ((pp3 & V_CTE & V_PTE & V_TPR & V_PEAP) -> EF ((s3 & vcd & vacuum) & V_CTE & V_PTE & V_TPR & V_PEAP))

--*****************************************
--PR_9 Quand le vérin vertical descend, tous les autres vérins sont et restent en fin de course

SPEC
     AG ((V_P2 & V_CTE & V_PTE & V_TPR & V_PEAC) -> ((V_P5 & V_P9) | (V_P5 & V_P11) | (V_P7 & V_P9) |
           (V_P7 & V_P11)))

SPEC
     AG ((V_P2 & V_CTE & V_PTE & V_TPR & V_PEAP) -> ((V_P5 & V_P9) | (V_P5 & V_P11) | (V_P7 & V_P9) |
           (V_P7 & V_P11)))

--*****************************************************
--******************** PROCESSUS**********************
--*****************************************************


--*********************************************
--VERTICAL CYLINDER ; FR=1

VAR
     VC : {P1, P2, P3, P4};
     FMT_VC : boolean;

DEFINE
     V_P1 := VC=P1;
     V_P2 := VC=P2;
     V_P3 := VC=P3;
     V_P4 := VC=P4;

ASSIGN
     init(VC) := P1;
     next(VC) := case
           VC=P1 & !VCGD & V_PTR & !FMT_VC & FR=1 : P1;
           VC=P1 & VCGD & V_PTR & !FMT_VC & FR=1 : P2;
           VC=P2 & !((VCGD & E1) | (!VCGD)) & V_PTR & !FMT_VC & FR=1 : P2;
           VC=P2 & VCGD & E1 & V_PTR & !FMT_VC & FR=1 : P3;
           VC=P2 & !VCGD & V_PTR & !FMT_VC & FR=1 : P4;
           VC=P3 & VCGD & V_PTR & !FMT_VC & FR=1 : P3;
           VC=P3 & !VCGD & V_PTR & !FMT_VC & FR=1 : P4;
           VC=P4 & !((!VCGD & E2) | (VCGD)) & V_PTR & !FMT_VC & FR=1  : P4;
           VC=P4 & !VCGD & E2 & V_PTR & !FMT_VC & FR=1  : P1;
           VC=P4 & VCGD & V_PTR & !FMT_VC & FR=1  : P2;
           1 : VC;
           esac;

     init(vcu) := 1;
     next(vcu) := case
           VC=P4 & !VCGD & E2 & V_PTR & !FMT_VC & FR=1 : 1;
           VC=P1 & VCGD & V_PTR & !FMT_VC & FR=1 : 0;
           1 : vcu;
           esac;

     init(vcd) := 0;
     next(vcd) := case
           VC=P2 & VCGD & E1 & V_PTR & !FMT_VC & FR=1 : 1;
           VC=P3 & !VCGD & V_PTR & !FMT_VC & FR=1 : 0;
           1 : vcd;
           esac;

     init(FMT_VC) := 0;
```

```
      next(FMT_VC) := case
            (VC=P1 & !VCGD & V_PTR & !FMT_VC & FR=1 ) |
            (VC=P1 & VCGD & V_PTR & !FMT_VC & FR=1) |
            (VC=P2 & !((VCGD & E1) | (!VCGD)) & V_PTR & !FMT_VC & FR=1) |
            (VC=P2 & VCGD & E1 & V_PTR & !FMT_VC & FR=1) |
            (VC=P2 & !VCGD & V_PTR & !FMT_VC & FR=1) |
            (VC=P3 & VCGD & V_PTR & !FMT_VC & FR=1) |
            (VC=P3 & !VCGD & V_PTR & !FMT_VC & FR=1) |
            (VC=P4 & !((!VCGD & E2) | (VCGD)) & V_PTR & !FMT_VC & FR=1) |
            (VC=P4 & !VCGD & E2 & V_PTR & !FMT_VC & FR=1) |
            (VC=P4 & VCGD & V_PTR & !FMT_VC & FR=1) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) |
            (SEQ_PLANT=PTR & CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                   & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3) : 0;
            1 : FMT_VC;
            esac;

--*********************************************
--L1 CYLINDER ; FR=2

VAR
      L1C : {P5, P6, P7, P8};
      FMT_L1C : boolean;

DEFINE
      V_P5 := L1C=P5;
      V_P6 := L1C=P6;
      V_P7 := L1C=P7;
      V_P8 := L1C=P8;

ASSIGN
      init(L1C) := P5;
      next(L1C) := case
            L1C=P5 & !(L1CGO & !L1CGI) & V_PTR & !FMT_L1C & FR=2  : P5;
            L1C=P5 & L1CGO & !L1CGI & V_PTR & !FMT_L1C & FR=2 : P6;
            L1C=P6 & !((!L1CGI & E3) | (L1CGI & !L1CGO)) & V_PTR & !FMT_L1C & FR=2 : P6;
            L1C=P6 & !L1CGI & E3 & V_PTR & !FMT_L1C & FR=2 : P7;
            L1C=P6 & L1CGI & !L1CGO & V_PTR & !FMT_L1C & FR=2 : P8;
            L1C=P7 & !(L1CGI & !L1CGO) & V_PTR & !FMT_L1C & FR=2 : P7;
            L1C=P7 & L1CGI & !L1CGO & V_PTR & !FMT_L1C & FR=2 : P8;
            L1C=P8 & !((!L1CGO & E4) | (L1CGO & !L1CGI)) & V_PTR & !FMT_L1C & FR=2 : P8;
            L1C=P8 & !L1CGO & E4 & V_PTR & !FMT_L1C & FR=2 : P5;
            L1C=P8 & L1CGO & !L1CGI & V_PTR & !FMT_L1C & FR=2  : P6;
            1 : L1C;
            esac;

      init(FMT_L1C) := 0;
      next(FMT_L1C) := case
            (L1C=P5 & !(L1CGO & !L1CGI) & V_PTR & !FMT_L1C & FR=2) |
            (L1C=P5 & L1CGO & !L1CGI & V_PTR & !FMT_L1C & FR=2) |
            (L1C=P6 & !((!L1CGI & E3) | (L1CGI & !L1CGO)) & V_PTR & !FMT_L1C & FR=2) |
            (L1C=P6 & !L1CGI & E3 & V_PTR & !FMT_L1C & FR=2) |
            (L1C=P6 & L1CGI & !L1CGO & V_PTR & !FMT_L1C & FR=2) |
            (L1C=P7 & !(L1CGI & !L1CGO) & V_PTR & !FMT_L1C & FR=2) |
            (L1C=P7 & L1CGI & !L1CGO & V_PTR & !FMT_L1C & FR=2) |
            (L1C=P8 & !((!L1CGO & E4) | (L1CGO & !L1CGI)) & V_PTR & !FMT_L1C & FR=2) |
            (L1C=P8 & !L1CGO & E4 & V_PTR & !FMT_L1C & FR=2) |
            (L1C=P8 & L1CGO & !L1CGI & V_PTR & !FMT_L1C & FR=2) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) |
            (SEQ_PLANT=PTR & CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                   & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3) : 0;
            1 : FMT_L1C;
            esac;

--*********************************************
--L2 CYLINDER ; FR=3

VAR
      L2C : {P9, P10, P11, P12};
      FMT_L2C : boolean;

DEFINE
      V_P9 := L2C=P9;
      V_P10 := L2C=P10;
      V_P11 := L2C=P11;
      V_P12 := L2C=P12;

ASSIGN
      init(L2C) := P9;
      next(L2C) := case
            L2C=P9 & !(L2CGO & !L2CGI) & V_PTR & !FMT_L2C & FR=3 : P9;
            L2C=P9 & L2CGO & !L2CGI & V_PTR & !FMT_L2C & FR=3 : P10;
            L2C=P10 & !((!L2CGI & E5) | (L2CGI & !L2CGO)) & V_PTR & !FMT_L2C & FR=3 : P10;
            L2C=P10 & !L2CGI & E5 & V_PTR & !FMT_L2C & FR=3 : P11;
            L2C=P10 & L2CGI & !L2CGO & V_PTR & !FMT_L2C & FR=3 : P12;
            L2C=P11 & !(L2CGI & !L2CGO) & V_PTR & !FMT_L2C & FR=3 : P11;
            L2C=P11 & L2CGI & !L2CGO & V_PTR & !FMT_L2C & FR=3 : P12;
```

```
            L2C=P12 & !((!L2CGO & E6) | (L2CGO & !L2CGI)) & V_PTR & !FMT_L2C & FR=3 : P12;
            L2C=P12 & !L2CGO & E6 & V_PTR & !FMT_L2C & FR=3  : P9;
            L2C=P12 & L2CGO & !L2CGI & V_PTR & !FMT_L2C & FR=3 : P10;
            1 : L2C;
            esac;

      init(FMT_L2C) := 0;
      next(FMT_L2C) := case
            (L2C=P9 & !(L2CGO & !L2CGI) & V_PTR & !FMT_L2C & FR=3) |
            (L2C=P9 & L2CGO & !L2CGI & V_PTR & !FMT_L2C & FR=3) |
            (L2C=P10 & !((!L2CGI & E5) | (L2CGI & !L2CGO)) & V_PTR & !FMT_L2C & FR=3) |
            (L2C=P10 & !L2CGI & E5 & V_PTR & !FMT_L2C & FR=3) |
            (L2C=P10 & L2CGI & !L2CGO & V_PTR & !FMT_L2C & FR=3) |
            (L2C=P11 & !(L2CGI & !L2CGO) & V_PTR & !FMT_L2C & FR=3) |
            (L2C=P11 & L2CGI & !L2CGO & V_PTR & !FMT_L2C & FR=3) |
            (L2C=P12 & !((!L2CGO & E6) | (L2CGO & !L2CGI)) & V_PTR & !FMT_L2C & FR=3) |
            (L2C=P12 & !L2CGO & E6 & V_PTR & !FMT_L2C & FR=3) |
            (L2C=P12 & L2CGO & !L2CGI & V_PTR & !FMT_L2C & FR=3) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) |
            (SEQ_PLANT=PTR & CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                  & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3) : 0;
            1 : FMT_L2C;
            esac;

--*********************************************************
--SENSOR s0 ; FR=4

VAR
      sensor_s0 : {P13, P14};
      FMT_s0p : boolean;

DEFINE
      V_P13 := sensor_s0=P13;
      V_P14 := sensor_s0=P14;

ASSIGN
      init(sensor_s0) := P14;
      next(sensor_s0) := case

            sensor_s0=P14 & V_P5 & V_P9 & V_PTR & !FMT_s0p & FR=4 : P14;
            sensor_s0=P14 & !(V_P5 & V_P9) & V_PTR & !FMT_s0p & FR=4 : P13;
            sensor_s0=P13 & !(V_P5 & V_P9) & V_PTR & !FMT_s0p & FR=4 : P13;
            sensor_s0=P13 & V_P5 & V_P9 & V_PTR & !FMT_s0p & FR=4 : P14;
            1 : sensor_s0;
            esac;

      init(s0) := 1;
      next(s0) := case
            sensor_s0=P13 & V_P5 & V_P9 & V_PTR & !FMT_s0p & FR=4 : 1;
            sensor_s0=P14 & !(V_P5 & V_P9) & V_PTR & !FMT_s0p & FR=4 : 0;
            1 : s0;
            esac;

      init(FMT_s0p) := 0;
      next(FMT_s0p) := case
            (sensor_s0=P13 & !(V_P5 & V_P9) & V_PTR & !FMT_s0p & FR=4) |
            (sensor_s0=P13 & V_P5 & V_P9 & V_PTR & !FMT_s0p & FR=4) |
            (sensor_s0=P14 & V_P5 & V_P9 & V_PTR & !FMT_s0p & FR=4) |
            (sensor_s0=P14 & !(V_P5 & V_P9) & V_PTR & !FMT_s0p & FR=4) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) |
            (SEQ_PLANT=PTR & CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                  & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3) : 0;
            1 : FMT_s0p;
            esac;

--*********************************************************
--SENSOR s1 ; FR=5

VAR
      sensor_s1 : {P15, P16};
      FMT_s1p : boolean;

DEFINE
      V_P15 := sensor_s1=P15;
      V_P16 := sensor_s1=P16;

ASSIGN
      init(sensor_s1) := P15;
      next(sensor_s1) := case
            sensor_s1=P15 & !(V_P7 & V_P9) & V_PTR & !FMT_s1p & FR=5 : P15;
            sensor_s1=P15 & V_P7 & V_P9 & V_PTR & !FMT_s1p & FR=5 : P16;
                        sensor_s1=P16 & V_P7 & V_P9 & V_PTR & !FMT_s1p & FR=5 : P16;
            sensor_s1=P16 & !(V_P7 & V_P9) & V_PTR & !FMT_s1p & FR=5 : P15;
            1 : sensor_s1;
            esac;

      init(s1) := 0;
```

```
      next(s1) := case
            sensor_s1=P15 & V_P7 & V_P9 & V_PTR & !FMT_s1p & FR=5 : 1;
            sensor_s1=P16 & !(V_P7 & V_P9) & V_PTR & !FMT_s1p & FR=5 : 0;
            1 : s1;
            esac;

      init(FMT_s1p) := 0;
      next(FMT_s1p) := case
            (sensor_s1=P15 & !(V_P7 & V_P9) & V_PTR & !FMT_s1p & FR=5) |
            (sensor_s1=P15 & V_P7 & V_P9 & V_PTR & !FMT_s1p & FR=5) |
            (sensor_s1=P16 & V_P7 & V_P9 & V_PTR & !FMT_s1p & FR=5) |
            (sensor_s1=P16 & !(V_P7 & V_P9) & V_PTR & !FMT_s1p & FR=5) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) |
            (SEQ_PLANT=PTR & CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                  & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3) : 0;
            1 : FMT_s1p;
            esac;

--*********************************************************
--SENSOR s2 ; FR=6

VAR
      sensor_s2 : {P17, P18};
      FMT_s2p : boolean;

DEFINE
      V_P17 := sensor_s2=P17;
      V_P18 := sensor_s2=P18;

ASSIGN
      init(sensor_s2) := P17;
      next(sensor_s2) := case
            sensor_s2=P17 & !(V_P5 & V_P11) & V_PTR & !FMT_s2p & FR=6 : P17;
            sensor_s2=P17 & V_P5 & V_P11 & V_PTR & !FMT_s2p & FR=6 : P18;
            sensor_s2=P18 & V_P5 & V_P11 & V_PTR & !FMT_s2p & FR=6 : P18;
            sensor_s2=P18 & !(V_P5 & V_P11) & V_PTR & !FMT_s2p & FR=6 : P17;
            1 : sensor_s2;
            esac;

      init(s2) := 0;
      next(s2) := case
            sensor_s2=P17 & V_P5 & V_P11 & V_PTR & !FMT_s2p & FR=6 : 1;
            sensor_s2=P18 & !(V_P5 & V_P11) & V_PTR & !FMT_s2p & FR=6 : 0;
            1 : s2;
            esac;

      init(FMT_s2p) := 0;
      next(FMT_s2p) := case
            (sensor_s2=P17 & !(V_P5 & V_P11) & V_PTR & !FMT_s2p & FR=6) |
            (sensor_s2=P17 & V_P5 & V_P11 & V_PTR & !FMT_s2p & FR=6) |
            (sensor_s2=P18 & V_P5 & V_P11 & V_PTR & !FMT_s2p & FR=6) |
            (sensor_s2=P18 & !(V_P5 & V_P11) & V_PTR & !FMT_s2p & FR=6) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) |
            (SEQ_PLANT=PTR & CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                  & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3) : 0;
            1 : FMT_s2p;
            esac;

--*********************************************************
--SENSOR s3 ; FR=7

VAR
      sensor_s3 : {P19, P20};
      FMT_s3p : boolean;

DEFINE
      V_P19 := sensor_s3=P19;
      V_P20 := sensor_s3=P20;

ASSIGN
      init(sensor_s3) := P19;
      next(sensor_s3) := case
            sensor_s3=P19 & !(V_P7 & V_P11) & V_PTR & !FMT_s3p & FR=7 : P19;
            sensor_s3=P19 & V_P7 & V_P11 & V_PTR & !FMT_s3p & FR=7 : P20;
            sensor_s3=P20 & V_P7 & V_P11 & V_PTR & !FMT_s3p & FR=7 : P20;
            sensor_s3=P20 & !(V_P7 & V_P11) & V_PTR & !FMT_s3p & FR=7 : P19;
            1 : sensor_s3;
            esac;

      init(s3) := 0;
      next(s3) := case
            sensor_s3=P19 & V_P7 & V_P11 & V_PTR & !FMT_s3p & FR=7 : 1;
            sensor_s3=P20 & !(V_P7 & V_P11) & V_PTR & !FMT_s3p & FR=7 : 0;
            1 : s3;
            esac;
```

```
      init(FMT_s3p) := 0;
      next(FMT_s3p) := case
            (sensor_s3=P19 & !(V_P7 & V_P11) & V_PTR & !FMT_s3p & FR=7) |
            (sensor_s3=P19 & V_P7 & V_P11 & V_PTR & !FMT_s3p & FR=7) |
            (sensor_s3=P20 & V_P7 & V_P11 & V_PTR & !FMT_s3p & FR=7) |
            (sensor_s3=P20 & !(V_P7 & V_P11) & V_PTR & !FMT_s3p & FR=7) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) |
            (SEQ_PLANT=PTR & CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                   & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3) : 0;
            1 : FMT_s3p;
            esac;

--*******************************************************
--ZONE V ; FR=8

VAR
      ZONE_V : {P21, P22};
      FMT_ZV : boolean;

DEFINE
      V_P21 := ZONE_V=P21;
      V_P22 := ZONE_V=P22;

ASSIGN
      init(ZONE_V) := P21;
      next(ZONE_V) := case
            ZONE_V=P21 & !(VENTURI & V_P3 & ((V_P7 & V_P9 & V_P27) |
                  (V_P5 & V_P11 & V_P30) | (V_P7 & V_P11 & V_P33)))
                  & V_PTR & !FMT_ZV & FR=8 : P21;
            ZONE_V=P21 & VENTURI & V_P3 & ((V_P7 & V_P9 & V_P27) | (V_P5 & V_P11 & V_P30) |
                  (V_P7 & V_P11 & V_P33)) & V_PTR & !FMT_ZV & FR=8 : P22;
            ZONE_V=P22 & VENTURI & V_PTR & !FMT_ZV  & FR=8 : P22;
            ZONE_V=P22 & !VENTURI & V_PTR & !FMT_ZV  & FR=8 : P21;
            1 : ZONE_V;
            esac;

      init(vacuum) := 0;
      next(vacuum) := case
            ZONE_V=P21 & VENTURI & V_P3 & ((V_P7 & V_P9 & V_P27) | (V_P5 & V_P11 & V_P30) |
                  (V_P7 & V_P11 & V_P33)) & V_PTR & !FMT_ZV & FR=8 : 1;
            ZONE_V=P22 & !VENTURI & V_PTR & !FMT_ZV  & FR=8 : 0;
            1 : vacuum;
            esac;

      init(FMT_ZV) := 0;
      next(FMT_ZV) := case
            (ZONE_V=P21 & !(VENTURI & V_P3 & ((V_P7 & V_P9 & V_P27) |
                  (V_P5 & V_P11 & V_P30) | (V_P7 & V_P11 & V_P33)))
                  & V_PTR & !FMT_ZV & FR=8) |
            (ZONE_V=P21 & VENTURI & V_P3 & ((V_P7 & V_P9 & V_P27) | (V_P5 & V_P11 & V_P30) |
                  (V_P7 & V_P11 & V_P33)) & V_PTR & !FMT_ZV & FR=8) |
            (ZONE_V=P22 & VENTURI & V_PTR & !FMT_ZV  & FR=8) |
            (ZONE_V=P22 & !VENTURI & V_PTR & !FMT_ZV  & FR=8) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) |
            (SEQ_PLANT=PTR & CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                   & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3) : 0;
            1 : FMT_ZV;
            esac;

--*******************************************************
--ZONE 0 ; FR=9

VAR
      ZONE_0 : {P23, P24, P25};
      FMT_Z0 : boolean;

DEFINE
      V_P23 := ZONE_0=P23;
      V_P24 := ZONE_0=P24;
      V_P25 := ZONE_0=P25;

ASSIGN
      init(ZONE_0) := P23;
      next(ZONE_0) := case
            ZONE_0=P23 & !(V_P3 & V_P5 & V_P9 & V_P22) & V_PTR & !FMT_Z0 & FR=9 : P23;
            ZONE_0=P23 & V_P3 & V_P5 & V_P9 & V_P22 & V_PTR & !FMT_Z0 & FR=9 : P24;
            ZONE_0=P24 & !(V_P21 & V_P4) & V_PTR & !FMT_Z0 & FR=9 : P24;
            ZONE_0=P24 & V_P21 & V_P4 & V_PTR & !FMT_Z0 & FR=9 : P25;
            ZONE_0=P25 & !E7 & V_PTR & !FMT_Z0 & FR=9 : P25;
            ZONE_0=P25 & E7 & V_PTR & !FMT_Z0 & FR=9 : P23;
            1 : ZONE_0;
            esac;

      init(pp0) := 0;
      next(pp0) := case
            ZONE_0=P23 & V_P3 & V_P5 & V_P9 & V_P22 & V_PTR & !FMT_Z0 & FR=9 : 1;
            ZONE_0=P24 & V_P21 & V_P4 & V_PTR & !FMT_Z0 & FR=9 : 0;
```

```
            1 : pp0;
            esac;

      init(FMT_Z0) := 0;
      next(FMT_Z0) := case
            (ZONE_0=P23 & !(V_P3 & V_P5 & V_P9 & V_P22) & V_PTR & !FMT_Z0 & FR=9) |
            (ZONE_0=P23 & V_P3 & V_P5 & V_P9 & V_P22 & V_PTR & !FMT_Z0 & FR=9) |
            (ZONE_0=P24 & !(V_P21 & V_P4) & V_PTR & !FMT_Z0 & FR=9) |
            (ZONE_0=P24 & V_P21 & V_P4 & V_PTR & !FMT_Z0 & FR=9) |
            (ZONE_0=P25 & !E7 & V_PTR & !FMT_Z0 & FR=9) |
            (ZONE_0=P25 & E7 & V_PTR & !FMT_Z0 & FR=9) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) |
            (SEQ_PLANT=PTR & CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                   & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3) : 0;
            1 : FMT_Z0;
            esac;

--*******************************************************
--ZONE 1 ; FR=10

VAR
      ZONE_1 : {P26, P27, P28};
      FMT_Z1 : boolean;

DEFINE
      V_P26 := ZONE_1=P26;
      V_P27 := ZONE_1=P27;
      V_P28 := ZONE_1=P28;

ASSIGN
      init(ZONE_1) := P26;
      next(ZONE_1) := case
            ZONE_1=P26 & !E8 & V_PTR & !FMT_Z1 & FR=10 : P26;
            ZONE_1=P26 & E8 & V_PTR & !FMT_Z1 & FR=10 : P27;
            ZONE_1=P27 & !(V_P4 & V_P7 & V_P9 & V_P22)  & V_PTR
                  & !FMT_Z1 & FR=10 : P27;
            ZONE_1=P27 & V_P4 & V_P7 & V_P9 & V_P22 & V_PTR
                  & !FMT_Z1 & FR=10 : P28;
            ZONE_1=P28 & !E9 & V_PTR & !FMT_Z1 & FR=10 : P28;
            ZONE_1=P28 & E9 & V_PTR & !FMT_Z1 & FR=10 : P26;
            1 : ZONE_1;
            esac;

      init(pp1) := 0;
      next(pp1) := case
            ZONE_1=P26 & E8 & V_PTR & !FMT_Z1 & FR=10 : 1;
            ZONE_1=P27 & V_P4 & V_P7 & V_P9 & V_P22 & V_PTR
                  & !FMT_Z1 & FR=10 : 0;
            1 : pp1;
            esac;

      init(FMT_Z1) := 0;
      next(FMT_Z1) := case
            (ZONE_1=P26 & !E8 & V_PTR & !FMT_Z1 & FR=10) |
            (ZONE_1=P26 & E8 & V_PTR & !FMT_Z1 & FR=10) |
            (ZONE_1=P27 & !(V_P4 & V_P7 & V_P9 & V_P22)  & V_PTR
                  & !FMT_Z1 & FR=10) |
            (ZONE_1=P27 & V_P4 & V_P7 & V_P9 & V_P22 & V_PTR
                  & !FMT_Z1 & FR=10) |
            (ZONE_1=P28 & !E9 & V_PTR & !FMT_Z1 & FR=10) |
            (ZONE_1=P28 & E9 & V_PTR & !FMT_Z1 & FR=10) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) |
            (SEQ_PLANT=PTR & CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                   & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3) : 0;
            1 : FMT_Z1;
            esac;

--*************************************************************
--ZONE 2 ; FR=11

VAR
      ZONE_2 : {P29, P30, P31};
      FMT_Z2 : boolean;

DEFINE
      V_P29 := ZONE_2=P29;
      V_P30 := ZONE_2=P30;
      V_P31 := ZONE_2=P31;

ASSIGN
      init(ZONE_2) := P29;
      next(ZONE_2) := case
            ZONE_2=P29 & !E10 & V_PTR & !FMT_Z2 & FR=11 : P29;
            ZONE_2=P29 & E10 & V_PTR & !FMT_Z2 & FR=11 : P30;
            ZONE_2=P30 & !(V_P4 & V_P5 & V_P11 & V_P22) & V_PTR
                  & !FMT_Z2 & FR=11 : P30;
            ZONE_2=P30 & V_P4 & V_P5 & V_P11 & V_P22 & V_PTR
```

```
                & !FMT_Z2 & FR=11 : P31;
          ZONE_2=P31 & !E11 & V_PTR & !FMT_Z2 & FR=11 : P31;
          ZONE_2=P31 & E11 & V_PTR & !FMT_Z2 & FR=11 : P29;
          1 : ZONE_2;
          esac;

     init(pp2) := 0;
     next(pp2) := case
          ZONE_2=P29 & E10 & V_PTR & !FMT_Z2 & FR=11 : 1;
          ZONE_2=P30 & V_P4 & V_P5 & V_P11 & V_P22 & V_PTR
                & !FMT_Z2 & FR=11 : 0;
          1 : pp2;
          esac;

     init(FMT_Z2) := 0;
     next(FMT_Z2) := case
          (ZONE_2=P29 & !E10 & V_PTR & !FMT_Z2 & FR=11) |
          (ZONE_2=P29 & E10 & V_PTR & !FMT_Z2 & FR=11) |
          (ZONE_2=P30 & !(V_P4 & V_P5 & V_P11 & V_P22) & V_PTR
                & !FMT_Z2 & FR=11) |
          (ZONE_2=P30 & V_P4 & V_P5 & V_P11 & V_P22 & V_PTR
                & !FMT_Z2 & FR=11) |
          (ZONE_2=P31 & !E11 & V_PTR & !FMT_Z2 & FR=11) |
          (ZONE_2=P31 & E11 & V_PTR & !FMT_Z2 & FR=11) : 1;
          (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) |
          (SEQ_PLANT=PTR & CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                 & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3) : 0;
          1 : FMT_Z2;
          esac;

--************************************************************
--ZONE 3 ; FR=12

VAR
     ZONE_3 : {P32, P33, P34};
     FMT_Z3 : boolean;

DEFINE
     V_P32 := ZONE_3=P32;
     V_P33 := ZONE_3=P33;
     V_P34 := ZONE_3=P34;

ASSIGN
     init(ZONE_3) := P32;
     next(ZONE_3) := case
          ZONE_3=P32 & !E12 & V_PTR & !FMT_Z3 & FR=12 : P32;
          ZONE_3=P32 & E12 & V_PTR & !FMT_Z3 & FR=12 : P33;
          ZONE_3=P33 & !(V_P4 & V_P7 & V_P11 & V_P22) & V_PTR
                & !FMT_Z3 & FR=12 : P33;
          ZONE_3=P33 & V_P4 & V_P7 & V_P11 & V_P22 & V_PTR
                & !FMT_Z3 & FR=12 : P34;
          ZONE_3=P34 & !E13 & V_PTR & !FMT_Z3 & FR=12 : P34;
          ZONE_3=P34 & E13 & V_PTR & !FMT_Z3 & FR=12 : P32;
          1 : ZONE_3;
          esac;

     init(pp3) := 0;
     next(pp3) := case
          ZONE_3=P32 & E12 & V_PTR & !FMT_Z3 & FR=12 : 1;
          ZONE_3=P33 & V_P4 & V_P7 & V_P11 & V_P22 & V_PTR
                & !FMT_Z3 & FR=12 : 0;
          1 : pp3;
          esac;

     init(FMT_Z3) := 0;
     next(FMT_Z3) := case
          (ZONE_3=P32 & !E12 & V_PTR & !FMT_Z3 & FR=12) |
          (ZONE_3=P32 & E12 & V_PTR & !FMT_Z3 & FR=12) |
          (ZONE_3=P33 & !(V_P4 & V_P7 & V_P11 & V_P22) & V_PTR
                & !FMT_Z3 & FR=12) |
          (ZONE_3=P33 & V_P4 & V_P7 & V_P11 & V_P22 & V_PTR
                & !FMT_Z3 & FR=12) |
          (ZONE_3=P34 & !E13 & V_PTR & !FMT_Z3 & FR=12) |
          (ZONE_3=P34 & E13 & V_PTR & !FMT_Z3 & FR=12) : 1;
          (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) |
          (SEQ_PLANT=PTR & CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                 & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3) : 0;
          1 : FMT_Z3;
          esac;

--*********************************************
--*************** TEMPORISATIONS ***************

--*********************************************
-- E1 ; FR=1, FR=2

VAR
```

```
      EV1 : {off, loading, on};
      FMT_E1 : boolean;
      E1 : boolean;

ASSIGN
      init(EV1) := off;
      next(EV1) := case
            EV1=off & !V_P2 & V_PIR & !FMT_E1 & FR=1 : off;
            EV1=off & V_P2 & V_PIR & !FMT_E1 & FR=1 : loading;
            EV1=loading & V_P2 & V_PIR & !FMT_E1 & FR=2 : loading;
            EV1=loading & V_P2 & V_PIR & !FMT_E1 & FR=1 : on;
            EV1=loading & !V_P2 & V_PIR & !FMT_E1 & FR=1 : off;
            EV1=on & !V_P2 & V_PIR & !FMT_E1 & FR=1 : off;
            1 : EV1;
            esac;

      init(FMT_E1) := 0;
      next(FMT_E1) := case
            (EV1=off & !V_P2 & V_PIR & !FMT_E1 & FR=1) |
                  (EV1=off & V_P2 & V_PIR & !FMT_E1 & FR=1) |
                  (EV1=loading & V_P2 & V_PIR & !FMT_E1 & FR=2) |
                  (EV1=loading & V_P2 & V_PIR & !FMT_E1 & FR=1) |
                  (EV1=loading & !V_P2 & V_PIR & !FMT_E1 & FR=1) |
                  (EV1=on & !V_P2 & V_PIR & !FMT_E1 & FR=1) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) : 0;
            1 : FMT_E1;
            esac;

      init(E1) := 0;
      next(E1) := case
            (EV1=loading & V_P2 & V_PIR & !FMT_E1 & FR=1) : 1;
            (EV1=on & !V_P2 & V_PIR & !FMT_E1 & FR=1) : 0;
            1 : E1;
            esac;

--************************************************
-- E2 : FR=3, FR=4

VAR
      EV2 : {off, loading, on};
      FMT_E2 : boolean;
      E2 : boolean;

ASSIGN
      init(EV2) := off;
      next(EV2) := case
            EV2=off & !V_P4 & V_PIR & !FMT_E2 & FR=3 : off;
            EV2=off & V_P4 & V_PIR & !FMT_E2 & FR=3 : loading;
            EV2=loading & V_P4 & V_PIR & !FMT_E2 & FR=4 : loading;
            EV2=loading & V_P4 & V_PIR & !FMT_E2 & FR=3 : on;
            EV2=loading & !V_P4 & V_PIR & !FMT_E2 & FR=3 : off;
            EV2=on & !V_P4 & V_PIR & !FMT_E2 & FR=3 : off;
            1 : EV2;
            esac;

      init(FMT_E2) := 0;
      next(FMT_E2) := case
            (EV2=off & !V_P4 & V_PIR & !FMT_E2 & FR=3) |
                  (EV2=off & V_P4 & V_PIR & !FMT_E2 & FR=3) |
                  (EV2=loading & V_P4 & V_PIR & !FMT_E2 & FR=4) |
                  (EV2=loading & V_P4 & V_PIR & !FMT_E2 & FR=3) |
                  (EV2=loading & !V_P4 & V_PIR & !FMT_E2 & FR=3) |
                  (EV2=on & !V_P4 & V_PIR & !FMT_E2 & FR=3) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) : 0;
            1 : FMT_E2;
            esac;

      init(E2) := 0;
      next(E2) := case
            (EV2=loading & V_P4 & V_PIR & !FMT_E2 & FR=3) : 1;
            (EV2=on & !V_P4 & V_PIR & !FMT_E2 & FR=3) : 0;
            1 : E2;
            esac;

--************************************************
--E3 ; FR=5, FR=6

VAR
      EV3 : {off, loading, on};
      FMT_E3 : boolean;
      E3 : boolean;

ASSIGN
      init(EV3) := off;
      next(EV3) := case
            EV3=off & !V_P6 & V_PIR & !FMT_E3 & FR=5 : off;
            EV3=off & V_P6 & V_PIR & !FMT_E3 & FR=5 : loading;
```

```
            EV3=loading & V_P6 & V_PIR & !FMT_E3 & FR=6 : loading;
            EV3=loading & V_P6 & V_PIR & !FMT_E3 & FR=5 : on;
            EV3=loading & !V_P6 & V_PIR & !FMT_E3 & FR=5 : off;
            EV3=on & !V_P6 & V_PIR & !FMT_E3 & FR=5 : off;
            1 : EV3;
            esac;

      init(FMT_E3) := 0;
      next(FMT_E3) := case
            (EV3=off & !V_P6 & V_PIR & !FMT_E3 & FR=5) |
                  (EV3=off & V_P6 & V_PIR & !FMT_E3 & FR=5) |
                  (EV3=loading & V_P6 & V_PIR & !FMT_E3 & FR=6) |
                  (EV3=loading & V_P6 & V_PIR & !FMT_E3 & FR=5) |
                  (EV3=loading & !V_P6 & V_PIR & !FMT_E3 & FR=5) |
                  (EV3=on & !V_P6 & V_PIR & !FMT_E3 & FR=5) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) : 0;
            1 : FMT_E3;
            esac;

      init(E3) := 0;
      next(E3) := case
            (EV3=loading & V_P6 & V_PIR & !FMT_E3 & FR=5) : 1;
            (EV3=on & !V_P6 & V_PIR & !FMT_E3 & FR=5) : 0;
            1 : E3;
            esac;


--*********************************************
--E4 ; FR=7, FR=8

VAR
      EV4 : {off, loading, on};
      FMT_E4 : boolean;
      E4 : boolean;

ASSIGN
      init(EV4) := off;
      next(EV4) := case
            EV4=off & !V_P8 & V_PIR & !FMT_E4 & FR=7 : off;
            EV4=off & V_P8 & V_PIR & !FMT_E4 & FR=7 : loading;
            EV4=loading & V_P8 & V_PIR & !FMT_E4 & FR=8 : loading;
            EV4=loading & V_P8 & V_PIR & !FMT_E4 & FR=7 : on;
            EV4=loading & !V_P8 & V_PIR & !FMT_E4 & FR=7 : off;
            EV4=on & !V_P8 & V_PIR & !FMT_E4 & FR=7 : off;
            1 : EV4;
            esac;

      init(FMT_E4) := 0;
      next(FMT_E4) := case
            (EV4=off & !V_P8 & V_PIR & !FMT_E4 & FR=7) |
                  (EV4=off & V_P8 & V_PIR & !FMT_E4 & FR=7) |
                  (EV4=loading & V_P8 & V_PIR & !FMT_E4 & FR=8) |
                  (EV4=loading & V_P8 & V_PIR & !FMT_E4 & FR=7) |
                  (EV4=loading & !V_P8 & V_PIR & !FMT_E4 & FR=7) |
                  (EV4=on & !V_P8 & V_PIR & !FMT_E4 & FR=7) : 1;

            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) : 0;
            1 : FMT_E4;
            esac;

      init(E4) := 0;
      next(E4) := case
            (EV4=loading & V_P8 & V_PIR & !FMT_E4 & FR=7) : 1;
            (EV4=on & !V_P8 & V_PIR & !FMT_E4 & FR=7) : 0;
            1 : E4;
            esac;

--*********************************************
--E5 : FR=9, FR=10

VAR
      EV5 : {off, loading, on};
      FMT_E5 : boolean;
      E5 : boolean;

ASSIGN
      init(EV5) := off;
      next(EV5) := case
            EV5=off & !V_P10 & V_PIR & !FMT_E5 & FR=9 : off;
            EV5=off & V_P10 & V_PIR & !FMT_E5 & FR=9 : loading;
            EV5=loading & V_P10 & V_PIR & !FMT_E5 & FR=10 : loading;
            EV5=loading & V_P10 & V_PIR & !FMT_E5 & FR=9 : on;
            EV5=loading & !V_P10 & V_PIR & !FMT_E5 & FR=9 : off;
            EV5=on & !V_P10 & V_PIR & !FMT_E5 & FR=9 : off;
            1 : EV5;
            esac;
```

```
      init(FMT_E5) := 0;
      next(FMT_E5) := case
            (EV5=off & !V_P10 & V_PIR & !FMT_E5 & FR=9) |
                  (EV5=off & V_P10 & V_PIR & !FMT_E5 & FR=9) |
                  (EV5=loading & V_P10 & V_PIR & !FMT_E5 & FR=10) |
                  (EV5=loading & V_P10 & V_PIR & !FMT_E5 & FR=9) |
                  (EV5=loading & !V_P10 & V_PIR & !FMT_E5 & FR=9) |
                  (EV5=on & !V_P10 & V_PIR & !FMT_E5 & FR=9) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) : 0;
            1 : FMT_E5;
            esac;

      init(E5) := 0;
      next(E5) := case
            (EV5=loading & V_P10 & V_PIR & !FMT_E5 & FR=9) : 1;
            (EV5=on & !V_P10 & V_PIR & !FMT_E5 & FR=9) : 0;
            1 : E5;
            esac;

--*********************************************
--E6 ; FR=11, FR=12

VAR
      EV6 : {off, loading, on};
      FMT_E6 : boolean;
      E6 : boolean;

ASSIGN
      init(EV6) := off;
      next(EV6) := case
            EV6=off & !V_P12 & V_PIR & !FMT_E6 & FR=11 : off;
            EV6=off & V_P12 & V_PIR & !FMT_E6 & FR=11 : loading;
            EV6=loading & V_P12 & V_PIR & !FMT_E6 & FR=12 : loading;
            EV6=loading & V_P12 & V_PIR & !FMT_E6 & FR=11 : on;
            EV6=loading & !V_P12 & V_PIR & !FMT_E6 & FR=11 : off;
            EV6=on & !V_P12 & V_PIR & !FMT_E6 & FR=11 : off;
            1 : EV6;
            esac;

      init(FMT_E6) := 0;
      next(FMT_E6) := case
            (EV6=off & !V_P12 & V_PIR & !FMT_E6 & FR=11) |
                  (EV6=off & V_P12 & V_PIR & !FMT_E6 & FR=11) |
                  (EV6=loading & V_P12 & V_PIR & !FMT_E6 & FR=12) |
                  (EV6=loading & V_P12 & V_PIR & !FMT_E6 & FR=11) |
                  (EV6=loading & !V_P12 & V_PIR & !FMT_E6 & FR=11) |
                  (EV6=on & !V_P12 & V_PIR & !FMT_E6 & FR=11) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) : 0;
            1 : FMT_E6;
            esac;

      init(E6) := 0;
      next(E6) := case
            (EV6=loading & V_P12 & V_PIR & !FMT_E6 & FR=11) : 1;
            (EV6=on & !V_P12 & V_PIR & !FMT_E6 & FR=11) : 0;
            1 : E6;
            esac;

--*********************************************
--E7 ; FR=13, FR=14

VAR
      EV7 : {off, loading, on};
      FMT_E7 : boolean;
      E7 : boolean;

ASSIGN
      init(EV7) := off;
      next(EV7) := case
            EV7=off & !V_P25 & V_PIR & !FMT_E7 & FR=13 : off;
            EV7=off & V_P25 & V_PIR & !FMT_E7 & FR=13 : loading;
            EV7=loading & V_P25 & V_PIR & !FMT_E7 & FR=14 : loading;
            EV7=loading & V_P25 & V_PIR & !FMT_E7 & FR=13 : on;
            EV7=loading & !V_P25 & V_PIR & !FMT_E7 & FR=13 : off;
            EV7=on & !V_P25 & V_PIR & !FMT_E7 & FR=13 : off;
            1 : EV7;
            esac;

      init(FMT_E7) := 0;
      next(FMT_E7) := case
            (EV7=off & !V_P25 & V_PIR & !FMT_E7 & FR=13) |
                  (EV7=off & V_P25 & V_PIR & !FMT_E7 & FR=13) |
                  (EV7=loading & V_P25 & V_PIR & !FMT_E7 & FR=14) |
                  (EV7=loading & V_P25 & V_PIR & !FMT_E7 & FR=13) |
                  (EV7=loading & !V_P25 & V_PIR & !FMT_E7 & FR=13) |
                  (EV7=on & !V_P25 & V_PIR & !FMT_E7 & FR=13) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
```

```
                FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) : 0;
          1 : FMT_E7;
          esac;

     init(E7) := 0;
     next(E7) := case
          (EV7=loading & V_P25 & V_PIR & !FMT_E7 & FR=13) : 1;
          (EV7=on & !V_P25 & V_PIR & !FMT_E7 & FR=13) : 0;
          1 : E7;
          esac;

--*********************************************
--E8 ; FR=15, FR=16

VAR
     EV8 : {off, loading, on};
     FMT_E8 : boolean;
     E8 : boolean;

ASSIGN
     init(EV8) := off;
     next(EV8) := case
          EV8=off & !V_P26 & V_PIR & !FMT_E8 & FR=15 : off;
          EV8=off & V_P26 & V_PIR & !FMT_E8 & FR=15 : loading;
          EV8=loading & V_P26 & V_PIR & !FMT_E8 & FR=16 : loading;
          EV8=loading & V_P26 & V_PIR & !FMT_E8 & FR=15 : on;
          EV8=loading & !V_P26 & V_PIR & !FMT_E8 & FR=15 : off;
          EV8=on & !V_P26 & V_PIR & !FMT_E8 & FR=15 : off;
          1 : EV8;
          esac;

     init(FMT_E8) := 0;
     next(FMT_E8) := case
          (EV8=off & !V_P26 & V_PIR & !FMT_E8 & FR=15) |
               (EV8=off & V_P26 & V_PIR & !FMT_E8 & FR=15) |
               (EV8=loading & V_P26 & V_PIR & !FMT_E8 & FR=16) |
               (EV8=loading & V_P26 & V_PIR & !FMT_E8 & FR=15) |
               (EV8=loading & !V_P26 & V_PIR & !FMT_E8 & FR=15) |
               (EV8=on & !V_P26 & V_PIR & !FMT_E8 & FR=15) : 1;
          (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
               FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) : 0;
          1 : FMT_E8;
          esac;

     init(E8) := 0;
     next(E8) := case
          (EV8=loading & V_P26 & V_PIR & !FMT_E8 & FR=15) : 1;
          (EV8=on & !V_P26 & V_PIR & !FMT_E8 & FR=15) : 0;
          1 : E8;
          esac;

--*********************************************
--E9 ; FR=17, FR=18

VAR
     EV9 : {off, loading, on};
     FMT_E9 : boolean;
     E9 : boolean;

ASSIGN
     init(EV9) := off;
     next(EV9) := case
          EV9=off & !V_P28 & V_PIR & !FMT_E9 & FR=17 : off;
          EV9=off & V_P28 & V_PIR & !FMT_E9 & FR=17 : loading;
          EV9=loading & V_P28 & V_PIR & !FMT_E9 & FR=18 : loading;
          EV9=loading & V_P28 & V_PIR & !FMT_E9 & FR=17 : on;
          EV9=loading & !V_P28 & V_PIR & !FMT_E9 & FR=17 : off;
          EV9=on & !V_P28 & V_PIR & !FMT_E9 & FR=17 : off;
          1 : EV9;
          esac;

     init(FMT_E9) := 0;
     next(FMT_E9) := case
          (EV9=off & !V_P28 & V_PIR & !FMT_E9 & FR=17) |
               (EV9=off & V_P28 & V_PIR & !FMT_E9 & FR=17) |
               (EV9=loading & V_P28 & V_PIR & !FMT_E9 & FR=18) |
               (EV9=loading & V_P28 & V_PIR & !FMT_E9 & FR=17) |
               (EV9=loading & !V_P28 & V_PIR & !FMT_E9 & FR=17) |
               (EV9=on & !V_P28 & V_PIR & !FMT_E9 & FR=17) : 1;
          (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
               FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) : 0;
          1 : FMT_E9;
          esac;

     init(E9) := 0;
     next(E9) := case
          (EV9=loading & V_P28 & V_PIR & !FMT_E9 & FR=17) : 1;
          (EV9=on & !V_P28 & V_PIR & !FMT_E9 & FR=17) : 0;
          1 : E9;
```

```
            esac;

--*************************************************
--E10 ; FR=19, FR=20

VAR
      EV10 : {off, loading, on};
      FMT_E10 : boolean;
      E10 : boolean;

ASSIGN
      init(EV10) := off;
      next(EV10) := case
            EV10=off & !V_P29 & V_PIR & !FMT_E10 & FR=19 : off;
            EV10=off & V_P29 & V_PIR & !FMT_E10 & FR=19 : loading;
            EV10=loading & V_P29 & V_PIR & !FMT_E10 & FR=20 : loading;
            EV10=loading & V_P29 & V_PIR & !FMT_E10 & FR=19 : on;
            EV10=loading & !V_P29 & V_PIR & !FMT_E10 & FR=19 : off;
            EV10=on & !V_P29 & V_PIR & !FMT_E10 & FR=19 : off;
            1 : EV10;
            esac;

      init(FMT_E10) := 0;
      next(FMT_E10) := case
            (EV10=off & !V_P29 & V_PIR & !FMT_E10 & FR=19) |
                  (EV10=off & V_P29 & V_PIR & !FMT_E10 & FR=19) |
                  (EV10=loading & V_P29 & V_PIR & !FMT_E10 & FR=20) |
                  (EV10=loading & V_P29 & V_PIR & !FMT_E10 & FR=19) |
                  (EV10=loading & !V_P29 & V_PIR & !FMT_E10 & FR=19) |
                  (EV10=on & !V_P29 & V_PIR & !FMT_E10 & FR=19) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) : 0;
            1 : FMT_E10;
            esac;

      init(E10) := 0;
      next(E10) := case
            (EV10=loading & V_P29 & V_PIR & !FMT_E10 & FR=19) : 1;
            (EV10=on & !V_P29 & V_PIR & !FMT_E10 & FR=19) : 0;
            1 : E10;
            esac;

--*************************************************
--E11 ; FR=21, FR=22

VAR
      EV11 : {off, loading, on};
      FMT_E11 : boolean;
      E11 : boolean;

ASSIGN
      init(EV11) := off;
      next(EV11) := case
            EV11=off & !V_P31 & V_PIR & !FMT_E11 & FR=21 : off;
            EV11=off & V_P31 & V_PIR & !FMT_E11 & FR=21 : loading;
            EV11=loading & V_P31 & V_PIR & !FMT_E11 & FR=22 : loading;
            EV11=loading & V_P31 & V_PIR & !FMT_E11 & FR=21 : on;
            EV11=loading & !V_P31 & V_PIR & !FMT_E11 & FR=21 : off;
            EV11=on & !V_P31 & V_PIR & !FMT_E11 & FR=21 : off;
            1 : EV11;
            esac;

      init(FMT_E11) := 0;
      next(FMT_E11) := case
            (EV11=off & !V_P31 & V_PIR & !FMT_E11 & FR=21) |
                  (EV11=off & V_P31 & V_PIR & !FMT_E11 & FR=21) |
                  (EV11=loading & V_P31 & V_PIR & !FMT_E11 & FR=22) |
                  (EV11=loading & V_P31 & V_PIR & !FMT_E11 & FR=21) |
                  (EV11=loading & !V_P31 & V_PIR & !FMT_E11 & FR=21) |
                  (EV11=on & !V_P31 & V_PIR & !FMT_E11 & FR=21) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) : 0;
            1 : FMT_E11;
            esac;

      init(E11) := 0;
      next(E11) := case
            (EV11=loading & V_P31 & V_PIR & !FMT_E11 & FR=21) : 1;
            (EV11=on & !V_P31 & V_PIR & !FMT_E11 & FR=21) : 0;
            1 : E11;
            esac;

--*************************************************
--E12 ; FR=23, FR=24

VAR
      EV12 : {off, loading, on};
      FMT_E12 : boolean;
      E12 : boolean;
```

```
ASSIGN
      init(EV12) := off;
      next(EV12) := case
            EV12=off & !V_P32 & V_PIR & !FMT_E12 & FR=23 : off;
            EV12=off & V_P32 & V_PIR & !FMT_E12 & FR=23 : loading;
            EV12=loading & V_P32 & V_PIR & !FMT_E12 & FR=24 : loading;
            EV12=loading & V_P32 & V_PIR & !FMT_E12 & FR=23 : on;
            EV12=loading & !V_P32 & V_PIR & !FMT_E12 & FR=23 : off;
            EV12=on & !V_P32 & V_PIR & !FMT_E12 & FR=23 : off;
            1 : EV12;
            esac;

      init(FMT_E12) := 0;
      next(FMT_E12) := case
            (EV12=off & !V_P32 & V_PIR & !FMT_E12 & FR=23) |
                  (EV12=off & V_P32 & V_PIR & !FMT_E12 & FR=23) |
                  (EV12=loading & V_P32 & V_PIR & !FMT_E12 & FR=24) |
                  (EV12=loading & V_P32 & V_PIR & !FMT_E12 & FR=23) |
                  (EV12=loading & !V_P32 & V_PIR & !FMT_E12 & FR=23) |
                  (EV12=on & !V_P32 & V_PIR & !FMT_E12 & FR=23) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) : 0;
            1 : FMT_E12;
            esac;

      init(E12) := 0;
      next(E12) := case
            (EV12=loading & V_P32 & V_PIR & !FMT_E12 & FR=23) : 1;
            (EV12=on & !V_P32 & V_PIR & !FMT_E12 & FR=23) : 0;
            1 : E12;
            esac;

--*************************************************
--E13 ; FR=25, FR=26

VAR
      EV13 : {off, loading, on};
      FMT_E13 : boolean;
      E13 : boolean;

ASSIGN
      init(EV13) := off;
      next(EV13) := case
            EV13=off & !V_P34 & V_PIR & !FMT_E13 & FR=25 : off;
            EV13=off & V_P34 & V_PIR & !FMT_E13 & FR=25 : loading;
            EV13=loading & V_P34 & V_PIR & !FMT_E13 & FR=26 : loading;
            EV13=loading & V_P34 & V_PIR & !FMT_E13 & FR=25 : on;
            EV13=loading & !V_P34 & V_PIR & !FMT_E13 & FR=25 : off;
            EV13=on & !V_P34 & V_PIR & !FMT_E13 & FR=25 : off;
            1 : EV13;
            esac;

      init(FMT_E13) := 0;
      next(FMT_E13) := case
            (EV13=off & !V_P34 & V_PIR & !FMT_E13 & FR=25) |
                  (EV13=off & V_P34 & V_PIR & !FMT_E13 & FR=25) |
                  (EV13=loading & V_P34 & V_PIR & !FMT_E13 & FR=26) |
                  (EV13=loading & V_P34 & V_PIR & !FMT_E13 & FR=25) |
                  (EV13=loading & !V_P34 & V_PIR & !FMT_E13 & FR=25) |
                  (EV13=on & !V_P34 & V_PIR & !FMT_E13 & FR=25) : 1;
            (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                  FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) : 0;
            1 : FMT_E13;
            esac;

      init(E13) := 0;
      next(E13) := case
            (EV13=loading & V_P34 & V_PIR & !FMT_E13 & FR=25) : 1;
            (EV13=on & !V_P34 & V_PIR & !FMT_E13 & FR=25) : 0;
            1 : E13;
            esac;

--*****************************************************
--******** VARIABLES DE RECHERCHE DE STABILITE *********
--*****************************************************

--*****************************************************
--CPL

VAR
      CPL : boolean;

ASSIGN
      init(CPL) := 0;
      next(CPL) := case

            (VC=P1 & VCGD & V_PTR & !FMT_VC & FR=1) |
                  (VC=P2 & VCGD & E1 & V_PTR & !FMT_VC & FR=1) |
```

```
                (VC=P2 & !VCGD & V_PTR & !FMT_VC & FR=1) |
                (VC=P3 & !VCGD & V_PTR & !FMT_VC & FR=1) |
                (VC=P4 & !VCGD & E2 & V_PTR & !FMT_VC & FR=1) |
                (VC=P4 & VCGD & V_PTR & !FMT_VC & FR=1) |

          (L1C=P5 & L1CGO & !L1CGI & V_PTR & !FMT_L1C & FR=2) |
                (L1C=P6 & !L1CGI & E3 & V_PTR & !FMT_L1C & FR=2) |
                (L1C=P6 & L1CGI & !L1CGO & V_PTR & !FMT_L1C & FR=2) |
                (L1C=P7 & L1CGI & !L1CGO & V_PTR & !FMT_L1C & FR=2) |
                (L1C=P8 & !L1CGO & E4 & V_PTR & !FMT_L1C & FR=2) |
                (L1C=P8 & L1CGO & !L1CGI & V_PTR & !FMT_L1C & FR=2) |

          (L2C=P9 & L2CGO & !L2CGI & V_PTR & !FMT_L2C & FR=3) |
                (L2C=P10 & !L2CGI & E5 & V_PTR & !FMT_L2C & FR=3) |
                (L2C=P10 & L2CGI & !L2CGO & V_PTR & !FMT_L2C & FR=3) |
                (L2C=P11 & L2CGI & !L2CGO & V_PTR & !FMT_L2C & FR=3) |
                (L2C=P12 & !L2CGO & E6 & V_PTR & !FMT_L2C & FR=3) |
                (L2C=P12 & L2CGO & !L2CGI & V_PTR & !FMT_L2C & FR=3) |

          (sensor_s0=P13 & V_P5 & V_P9 & V_PTR & !FMT_s0p & FR=4) |
                (sensor_s0=P14 & !(V_P5 & V_P9) & V_PTR & !FMT_s0p & FR=4) |

          (sensor_s1=P15 & V_P7 & V_P9 & V_PTR & !FMT_s1p & FR=5) |
                (sensor_s1=P16 & !(V_P7 & V_P9) & V_PTR & !FMT_s1p & FR=5) |

          (sensor_s2=P17 & V_P5 & V_P11 & V_PTR & !FMT_s2p & FR=6) |
                (sensor_s2=P18 & !(V_P5 & V_P11) & V_PTR & !FMT_s2p & FR=6) |

          (sensor_s3=P19 & V_P7 & V_P11 & V_PTR & !FMT_s3p & FR=7) |
                (sensor_s3=P20 & !(V_P7 & V_P11) & V_PTR & !FMT_s3p & FR=7) |

          (ZONE_V=P21 & VENTURI & V_P3 & ((V_P7 & V_P9 & V_P27) | (V_P5 & V_P11 & V_P30) |
                (V_P7 & V_P11 & V_P33)) & V_PTR & !FMT_ZV & FR=8) |
                (ZONE_V=P22 & !VENTURI & V_PTR & !FMT_ZV  & FR=8) |

          (ZONE_0=P23 & V_P3 & V_P5 & V_P9 & V_P22 & V_PTR & !FMT_Z0 & FR=9) |
                (ZONE_0=P24 & V_P21 & V_P4 & V_PTR & !FMT_Z0 & FR=9) |
                (ZONE_0=P25 & E7 & V_PTR & !FMT_Z0 & FR=9) |

          (ZONE_1=P26 & E8 & V_PTR & !FMT_Z1 & FR=10) |
                (ZONE_1=P27 & V_P4 & V_P7 & V_P9 & V_P22 & V_PTR & !FMT_Z1 & FR=10) |
                (ZONE_1=P28 & E9 & V_PTR & !FMT_Z1 & FR=10) |

          (ZONE_2=P29 & E10 & V_PTR & !FMT_Z2 & FR=11) |
                (ZONE_2=P30 & V_P4 & V_P5 & V_P11 & V_P22 & V_PTR & !FMT_Z2 & FR=11) |
                (ZONE_2=P31 & E11 & V_PTR & !FMT_Z2 & FR=11) |

          (ZONE_3=P32 & E12 & V_PTR & !FMT_Z3 & FR=12) |
                (ZONE_3=P33 & V_P4 & V_P7 & V_P11 & V_P22 & V_PTR & !FMT_Z3 & FR=12) |
                (ZONE_3=P34 & E13 & V_PTR & !FMT_Z3 & FR=12) : 1;

          (SEQ_PLANT=PIR & !V_ST_PLEV & FMT_E1 & FMT_E2 & FMT_E3 & FMT_E4 & FMT_E5 & FMT_E6 & FMT_E7 & FMT_E8 &
                FMT_E9 & FMT_E10 & FMT_E11 & FMT_E12 & FMT_E13) |
          (SEQ_PLANT=PTR & CPL & FMT_VC & FMT_L1C & FMT_L2C & FMT_s0p & FMT_s1p & FMT_s2p
                 & FMT_s3p & FMT_ZV & FMT_Z0 & FMT_Z1 & FMT_Z2 & FMT_Z3) : 0;
                1 : CPL;
                esac;

--*********************************************************************
--CPRL

VAR
     CPRL : boolean;

ASSIGN
     init(CPRL) := 0;
     next(CPRL) := case
          (PR_4=PR1 & ((s1 | s2 | s3) & vcd) & vacuum & V_DPR & !FPRT_PR_4) |
                (PR_4=PR2 & (s0 & vcd) & !vacuum & V_DPR & !FPRT_PR_4) |
                (PR_4=PR2 & !(s0 & vcd) & !vacuum & V_DPR & !FPRT_PR_4) |
                (PR_4=PR3 & ((s1 | s2 | s3) & vcd) & !vacuum & V_DPR & !FPRT_PR_4) : 1;

          (SEQ_PROP=SPR & !(V_ST_PEAC | V_ST_PEAP)) |
          (SEQ_PROP=DPR & CPRL & FPRT_PR_4) : 0;

                1 : CPRL;
                esac;
```

A2.2 Sorties de la commande Unix «time NuSMV -r -reorder -dynamic exemple_support-specif_OK_MB.nusmv»

```
*** This is NuSMV 2.1.2 (compiled 2002-11-22 12:00:00)
*** For more information of NuSMV see <http://nusmv.irst.itc.it>
*** or email to <nusmv-users@irst.itc.it>.
*** Please report bugs to <nusmv-users@irst.itc.it>.
-- specification AG (!((((((L1CGI & L1CGO) & V_CTE) & V_PTE) & V_TPR) & V_PEAC)) is true
-- specification AG (!((((((L1CGI & L1CGO) & V_CTE) & V_PTE) & V_TPR) & V_PEAP)) is true
-- specification AG (!((((((L2CGI & L2CGO) & V_CTE) & V_PTE) & V_TPR) & V_PEAC)) is true
-- specification AG (!((((((L2CGI & L2CGO) & V_CTE) & V_PTE) & V_TPR) & V_PEAP)) is true
-- specification AG (((((VCGD & V_CTE) & V_PTE) & V_TPR) & V_PEAC) -> !(((L1CGI | L1CGO) | L2CGI) |
L2CGO)) is true
-- specification AG (((((VCGD & V_CTE) & V_PTE) & V_TPR) & V_PEAP) -> !(((L1CGI | L1CGO) | L2CGI) |
L2CGO)) is true
-- specification AG (((((((L1CGI | L1CGO) | L2CGI) | L2CGO) & V_CTE) & V_PTE) & V_TPR) & V_PEAC) ->
 vcu) is true
-- specification AG (((((((L1CGI | L1CGO) | L2CGI) | L2CGO) & V_CTE) & V_PTE) & V_TPR) & V_PEAP) ->
 vcu) is true
-- specification AG (!V_PR3) is true
-- specification AG (((((((V_P6 | V_P8) | V_P10) | V_P12) & V_CTE) & V_PTE) & V_TPR) & V_PEAC) -> v
cu) is true
-- specification AG (((((((V_P6 | V_P8) | V_P10) | V_P12) & V_CTE) & V_PTE) & V_TPR) & V_PEAP) -> v
cu) is true
-- specification AG (X1 -> EF (!X1)) is true
-- specification AG (X2 -> EF (!X2)) is true
-- specification AG (X3 -> EF (!X3)) is true
-- specification AG (X4 -> EF (!X4)) is true
-- specification AG (X10 -> EF (!X10)) is true
-- specification AG (X11 -> EF (!X11)) is true
-- specification AG (X20 -> EF (!X20)) is true
-- specification AG (X21 -> EF (!X21)) is true
-- specification AG (X30 -> EF (!X30)) is true
-- specification AG (X31 -> EF (!X31)) is true
-- specification AG (X32 -> EF (!X32)) is true
-- specification AG (X33 -> EF (!X33)) is true
-- specification AG (X34 -> EF (!X34)) is true
-- specification AG (X35 -> EF (!X35)) is true
-- specification AG (X36 -> EF (!X36)) is true
-- specification AG (X37 -> EF (!X37)) is true
-- specification AG (X38 -> EF (!X38)) is true
-- specification AG (((((pp1 & V_CTE) & V_PTE) & V_TPR) & V_PEAC) -> EF ((((V_P6 & V_CTE) & V_PTE) &
 V_TPR) & V_PEAC)) is true
-- specification AG (((((pp1 & V_CTE) & V_PTE) & V_TPR) & V_PEAP) -> EF ((((V_P6 & V_CTE) & V_PTE) &
 V_TPR) & V_PEAP)) is true
-- specification AG (((((pp2 & V_CTE) & V_PTE) & V_TPR) & V_PEAC) -> EF ((((V_P10 & V_CTE) & V_PTE)
& V_TPR) & V_PEAC)) is true
-- specification AG (((((pp2 & V_CTE) & V_PTE) & V_TPR) & V_PEAP) -> EF ((((V_P10 & V_CTE) & V_PTE)
& V_TPR) & V_PEAP)) is true
-- specification AG (((((pp3 & V_CTE) & V_PTE) & V_TPR) & V_PEAC) -> EF (((((V_P6 & V_P10) & V_CTE)
& V_PTE) & V_TPR) & V_PEAC)) is true
-- specification AG (((((pp3 & V_CTE) & V_PTE) & V_TPR) & V_PEAP) -> EF (((((V_P6 & V_P10) & V_CTE)
& V_PTE) & V_TPR) & V_PEAP)) is true
-- specification AG (((((pp1 & V_CTE) & V_PTE) & V_TPR) & V_PEAC) -> EF ((((((s1 & vcd) & vacuum) &
V_CTE) & V_PTE) & V_TPR) & V_PEAC)) is true
-- specification AG (((((pp1 & V_CTE) & V_PTE) & V_TPR) & V_PEAP) -> EF ((((((s1 & vcd) & vacuum) &
V_CTE) & V_PTE) & V_TPR) & V_PEAP)) is true
-- specification AG (((((pp2 & V_CTE) & V_PTE) & V_TPR) & V_PEAC) -> EF ((((((s2 & vcd) & vacuum) &
V_CTE) & V_PTE) & V_TPR) & V_PEAC)) is true
-- specification AG (((((pp2 & V_CTE) & V_PTE) & V_TPR) & V_PEAP) -> EF ((((((s2 & vcd) & vacuum) &
V_CTE) & V_PTE) & V_TPR) & V_PEAP)) is true
-- specification AG (((((pp3 & V_CTE) & V_PTE) & V_TPR) & V_PEAC) -> EF ((((((s3 & vcd) & vacuum) &
V_CTE) & V_PTE) & V_TPR) & V_PEAC)) is true
-- specification AG (((((pp3 & V_CTE) & V_PTE) & V_TPR) & V_PEAP) -> EF ((((((s3 & vcd) & vacuum) &
V_CTE) & V_PTE) & V_TPR) & V_PEAP)) is true
-- specification AG ((((((V_P2 & V_CTE) & V_PTE) & V_TPR) & V_PEAC) -> ((((V_P5 & V_P9) | (V_P5 & V_P
11)) | (V_P7 & V_P9)) | (V_P7 & V_P11))) is true
-- specification AG ((((((V_P2 & V_CTE) & V_PTE) & V_TPR) & V_PEAC) -> ((((V_P5 & V_P9) | (V_P5 & V_P
11)) | (V_P7 & V_P9)) | (V_P7 & V_P11))) is true

========= starting reordering ============

========= after reordering ============
reachable states: 3.84536e+08 (2^28.5185) out of 3.89317e+47 (2^158.092)

real    561m9.820s
user    524m49.409s
sys     0m10.885s
```