CONCEPTION PAR COMPOSANTES DE CONTRÔLEURS D'USINES MODULAIRES UTILISANT LA THÉORIE DU CONTRÔLE SUPERVISÉ

par

Daniel Côté

thèse présentée au Département d'informatique en vue
de l'obtention du grade de docteur ès sciences (Ph.D.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, Juin 2011

Your file Votre référence
ISBN: 978-0-494-83278-3

Our file Notre référence
ISBN: 978-0-494-83278-3

Sommaire

La complexité croissante des processus industriels et de leurs systèmes de contrôle rend de plus en plus attrayant l'usage des méthodes formelles pour leur conception. Plusieurs méthodes ont été mises au point tant pour la synthèse que pour la vérification, utilisant divers formalismes pour la modélisation de problèmes de contrôle et le raisonnement. Lorsqu'un processus peut être modélisé sous forme d'un système à événements discrets, la théorie du contrôle supervisé, originalement formulée par Ramadge et Wonham, offre une base formelle intéressante pour la spécification de problèmes de contrôle car elle permet, par l'application de procédures de synthèse, d'obtenir automatiquement un contrôleur pour le processus. La théorie souffre cependant d'un problème d'explosion combinatoire puisqu'elle utilise des automates d'états finis comme formalisme de modélisation. Plusieurs investigations se sont concentrées sur les moyens de mitiger ce problème en prenant avantage, soit de la structure des systèmes modélisés, soit des propriétés de leur spécification. Il en résulte plusieurs formes de la théorie dont, entre autres, les variantes modulaire, répartie et hiérarchique. D'autre part, certains efforts de recherche se sont concentrés sur le problème de l'implémentation des contrôleurs obtenus par les procédures de synthèse de la théorie. Il existe donc à ce jour plusieurs implémentations de ces procédures couvrant toutes les variantes de la théorie. Mais il ne semble pas y avoir encore d'environnement couvrant le processus de conception au complet. Le problème est encore plus aigu si l'on considère que la phase de modélisation est en général mal définie, voire même ignorée, dans ces implémentations. La présente thèse se propose de dégager les principes permettant de concevoir un environnement couvrant l'ensemble du processus d'ingénierie de contrôleurs dans le cadre de la théorie du contrôle supervisé, intégrant un outillage adéquat pour les trois phases du processus : modélisation, synthèse et génération de code.

Le 10 juin 2011

le jury a accepté la thèse de Monsieur Daniel Côté
dans sa version finale.

Membres du jury

Professeur Richard St-Denis
Directeur de recherche
Département d'informatique

Professeur Ahmed Khoumsi
Évaluateur externe au programme
Département de génie électrique et de génie informatique
Faculté de génie

Professeur Jules Desharnais
Évaluateur externe
Département d'informatique et de génie logiciel
Université Laval

Professeur Marc Frappier
Président rapporteur
Département d'informatique

Remerciements

J'adresse mes remerciements les plus sincères à mon directeur de thèse M. Richard St-Denis de l'Université de Sherbrooke pour son soutien indéfectible. Non seulement a-t-il révisé le texte de cette thèse avec une constance et une minutie exemplaires mais il a souvent été une source de commentaires critiques très précieuse. Je le remercie aussi pour le soutien financier qu'il m'a apporté à même ses propres fonds de recherche.

Abréviations

AFD Automate fini déterministe.

BDD, IDD et MDD *Binary Decision Diagram* (voir [12]), *Integer Decision Diagram* (voir [26]) et *Multi-valued Decision Diagram* (voir [30]) ; structures de données de la famille des diagrammes de décision.

DSFBC De l'anglais *Dynamic State Feed-Back Control* ou *Dynamic SFBC* ; loi de contrôle de type SFBC (voir ci-après) dans laquelle on modélise explicitement les éléments de mémoire.

HISC De l'anglais *Hierarchical Interface-based Supervisory Control* une variante hiérarchique de la théorie du contrôle supervisé (voir [35, 38]).

PLC De l'anglais *Programmable Logic Controller* ; en français, *automate programmable.*

ROBDD et ROMDD *Reduced Ordered Binary Decision Diagram* (voir [12]) et *Reduced Ordered Multi-valued Decision Diagram* (voir [30]).

SCT De l'anglais *Supervisory Control Theory* ; théorie du contrôle supervisé.

SED Systèmes à événements discrets, en anglais *Discrete-Event Systems* et aussi parfois *Discrete Event Dynamic Systems.*

SFBC De l'anglais *State Feed-Back Control* ; loi de contrôle basée sur une fonction de rétroaction définie sur l'espace des états que peut prendre un système à événements discrets.

STS De l'anglais *State Tree Structures* ; structures arborescentes représentant des familles de *configurations* au sens donné par D. Harel et M. Politi dans [27] pour décrire les états d'un système représenté à l'aide de *statecharts.*

Table des matières

Liste des tableaux

Liste des figures

Introduction

Le cycle de vie de plus en plus court des produits manufacturés accentue davantage la nécessité d'utiliser des chaînes de montage modulaires contrôlées par logiciel et en conséquence la nécessité de produire rapidement des contrôleurs logiciels corrects et fiables [46], [22]. À cette fin l'utilisation des méthodes formelles dans les applications de contrôle de procédé peut procurer une aide appréciable car ces dernières se prêtent en général à la génération automatique de code.

Les méthodes formelles se présentent sous forme de modèles mathématiques des systèmes permettant d'appliquer des techniques de raisonnement ainsi que des transformations pour en effectuer la vérification ou la synthèse. La vérification réfère à une variété de techniques servant à prouver qu'un programme, donné *a priori*, satisfait un ensemble de propriétés. La synthèse utilise une série de procédures permettant la génération automatique, ou systématique, de programmes satisfaisant par construction un ensemble de propriétés. Ces deux types d'approche ont été appliqués à la réalisation de programmes pour automates programmables (par exemple [5] pour la vérification et [33] pour la synthèse).

La théorie du contrôle supervisé (SCT – de l'anglais *Supervisory Control Theory*) est une méthode formelle qui procure une base théorique intéressante pour la résolution de problèmes de contrôle applicables aux systèmes à événements discrets (SED) [45]. Un SED est caractérisé par un espace d'états, un ensemble fini d'événements et une dynamique qui s'exprime par occurrence d'événements discrets. On peut considérer qu'en fonctionnement un SED génère des chaînes d'événements appartenant à un langage et à toute fin pratique SCT modélise un SED par un automate d'états fini déterministe (AFD). Pour fin de contrôle l'*alphabet* d'événements est partitionné en deux sous-ensembles : les événements contrôlables dont on peut inhiber l'occurrence et les événements incontrôlables. Avec cet équipe-

ment formel il est possible de concevoir des contrôleurs *abstraits*, appelés ici *superviseurs*, qui exercent une action de contrôle par inhibition d'événements contrôlables.

SCT inclut plusieurs éléments permettant d'entrevoir la possibilité d'utiliser un environnement intégré de conception de contrôleurs pour les systèmes de contrôle de procédés. La modélisation des systèmes de contrôle de procédés sous forme de SED semble intuitivement adéquate. La théorie comprend des procédures de synthèse permettant la génération automatique de superviseurs corrects à partir de modèles formels du système et des requis de contrôle qui s'y appliquent. Ces procédures sont déjà implémentées pour plusieurs variantes de la théorie (par exemple *TCT* dans [62]). Cependant, bien que plusieurs efforts fructueux aient été consentis dans cette direction (voir, entre autres, *IDES* [47] et *Supremica* [2]), il n'existe encore aucun système intégré d'usage courant pour l'application de SCT à l'ingénierie des systèmes de contrôle de procédés. Il est utile de s'interroger sur cet état des faits.

La solution idéale à ce problème de transfert technologique passe probablement par la génération automatique de code et par des outils spécialisés utilisant des représentations formelles des systèmes et des requis de contrôle. Or Ekberg et Krogh dans [13] font ressortir que les notations et les concepts impliqués représentent souvent une difficulté trop abrupte pour les non-spécialistes de la théorie. Bien sûr l'expression des systèmes de contrôle sous forme de SED est connue et utilisée dans la pratique (bien qu'elle n'y soit pas prédominante). Mais la spécification des requis de contrôle sous forme d'AFD se révèle très ardue, particulièrement lorsque l'on tente d'aborder des problèmes d'une envergure réaliste (voir, par exemple, [37]). En termes de pratique, il y a donc un malaise par rapport au formalisme normalement utilisé en SCT pour la spécification des requis. Il y a lieu d'investiguer pour tenter d'identifier un formalisme plus adéquat sans compromettre la possibilité de synthèse.

L'expression des systèmes à contrôler par des AFD nécessite habituellement une phase de modélisation *ad hoc*. Il est en effet impossible de passer d'un dispositif physique à une représentation formelle sans avoir recours à un certain nombre d'hypothèses sur son fonctionnement (voir à ce sujet [33]). La théorie présume en général que tous les modèles sont obtenus de cette façon sans jamais poser la question de la *validité* des modèles ainsi obtenus. Or la qualité d'une synthèse dépend directement de la qualité des modèles qui lui sont soumis. Obtenir de bons modèles de façon *ad hoc* est un processus coûteux et on aurait

avantage à pouvoir réutiliser ces modèles. Il y a d'ailleurs en général beaucoup de redondance dans l'ensemble des composantes d'un système de contrôle (voir [13]). Le nombre de *types* de dispositifs distincts est souvent assez restreint mais le nombre d'*instances* de n'importe lequel de ces dispositifs dans un système peut être assez important. De plus la modélisation *ad hoc* a tendance à pousser les concepteurs à *adapter* leurs modèles au problème de contrôle à résoudre de sorte que, d'un concepteur à l'autre, deux modèles d'un même dispositif peuvent s'avérer très différents.

La formulation d'une métaphore de *composantes réutilisables*, par exemple, peut apporter un soulagement aux problèmes de la modélisation *ad hoc*. Elle procure la réutilisation de modèles *fondamentaux* obtenus de façon *ad hoc* à partir du matériel ce qui permet d'amortir les coûts d'un travail long et ardu mais inévitable. Elle peut procurer en même temps une façon de réutiliser (par encapsulation et modularisation) des requis de contrôle ce qui n'est pas non plus à négliger.

Les considérations relatives à la génération automatique de code sont en général absentes des documents de recherche. Cette situation est compréhensible puisqu'il est normalement présumé que cette phase de la recherche appartient au domaine des applications et non de la recherche fondamentale. Dans la perspective d'un environnement intégré de conception de contrôleurs SCT il est impossible d'ignorer ces considérations car il est possible qu'il soit nécessaire de modifier le formalisme de modélisation pour réaliser la génération de code. La génération de code requiert souvent l'usage de code d'intégration (*glue code*, voir [25] entre autres) ou d'autres mécanismes ayant une interaction avec les modèles abstraits. Il faut aussi considérer comment il est possible de faire usage des résultats de synthèse et mettre en oeuvre la dynamique des modèles abstraits utilisés dans la spécification.

L'argument de Ekberg et Krogh dans [13] est plus qu'une simple complainte. Il se présente sous forme d'une question : est-il possible pour un spécialiste de SCT de faire un rapprochement entre une technique utilisée dans la pratique, les *schémas* de machines à états (*State Machine Templates*), et le modèle d'un problème de contrôle en SCT ? Cette question est tout à fait pertinente. Intuitivement, les AFD utilisés en SCT pour formuler des problèmes de contrôle présentent plus qu'une ressemblance superficielle avec ces *schémas* de machines à états. Si un spécialiste de la théorie ne peut pas faire de rapprochement utile entre les deux, comment un non-spécialiste le pourrait-il ? On peut d'ailleurs se convaincre

assez facilement, par l'usage, que ce qui semble manquer aux tentatives actuelles est précisément un genre de scénario d'usage *standard* pour leur environnement. Tout semble construit en présumant que les usagers connaissent très bien SCT et que les modèles (tous construits *ad hoc*) sont *ajustés* pour l'usage de la théorie. Bien que *Supremica* contienne des exemples pertinents de conception, il est assez difficile d'identifier les requis de contrôle de la solution et d'en déduire des conditions de garde pour des *schémas* de machines à états. Chaque solution est *unique* et dépend presque entièrement de l'ingéniosité du concepteur et de sa familiarité avec la théorie.

Cette question peut sembler facile à répondre. Mais le fait que l'article de Ekberg et Krogh date de 2006 et ne semble pas avoir trouvé preneur jusqu'à maintenant est symptomatique. Ou bien les spécialistes ne connaissent pas la réponse, ou bien la réponse est trop compliquée à élaborer et à rendre dans des termes clairs et simples. Cette thèse se propose donc d'identifier, parmi les éléments présents dans SCT, ceux qui peuvent servir à l'élaboration d'une plateforme de conception de systèmes à événements discrets. Cet effort s'inscrit donc plus dans une perspective de transfert technologique que dans une perspective de recherche fondamentale. Autant que possible on y évite d'altérer la théorie, mais il est possible qu'en cours de route il soit nécessaire d'y ajouter certains résultats. Quoiqu'il en soit l'envergure de cette entreprise semble être digne d'une thèse de doctorat.

Objectifs et hypothèses de travail

Brièvement, la thèse vise à progresser vers un environnement de conception de contrôleurs SCT utilisant une métaphore de composantes pour permettre la réutilisation de spécifications et limiter le besoin de conception *ad hoc* des modèles. Cet environnement doit permettre de modéliser un système existant sur lequel on veut implémenter un certain processus à partir d'un ensemble de requis donnés dans un cahier de charges (une spécification informelle). L'environnement doit supporter un usager dans cette tâche jusqu'à l'obtention d'un système fonctionnel répondant à l'ensemble des requis.

Pour fixer les idées, il est utile de se donner un scénario d'usage typique. À partir du cahier de charges, un usager doit progresser par raffinements successifs, c'est-à-dire par assemblage et composition d'instances de composantes réutilisables, vers un système

complet répondant à l'ensemble des requis de contrôle. À chaque itération l'usager doit pouvoir :

- assembler un certain nombre de composantes ;
- spécifier l'ensemble des requis de contrôle qui s'y appliquent (à son avis, puisqu'il s'agit là d'un processus de réécriture des requis) ;
- puis pouvoir expérimenter avec le système dans un état d'élaboration partiel et se convaincre que l'expression des requis est satisfaisante.

En d'autres termes l'usager *progresse* vers une solution contrôlable (et correcte) du problème (le cahier de charges). Ce scénario est légèrement différent de ce qui est normalement rencontré avec SCT où la spécification est considérée correcte *a priori*. Il est d'ailleurs étonnant de constater que dans la littérature de recherche sur le sujet il est toujours présumé que la spécification formelle est correcte au départ.

Pour ne pas se perdre en généralités, il est préférable de se donner un contexte puisque les systèmes à modéliser ne sont pas abstraits. Il peut être présumé sans perte de généralité que les systèmes matériels sont des *usines modulaires*. Ce genre d'usine est composée de dispositifs mécaniques élémentaires (vérins pneumatiques, relais et autres) qui peuvent être assemblés et ré-assemblés au besoin pour former des dispositifs plus complexes et s'adapter à divers procédés de fabrication. Une usine modulaire est habituellement contrôlée à l'aide d'un automate programmable souvent désigné sous le vocable de PLC (de l'anglais pour *Programmable Logic Controller*). Il s'agit d'un micro-ordinateur spécialisé pour fonctionner en usine. On peut donc aussi présumer, sans perte de généralité, que l'un des objectifs est de générer du code pour PLC.

À la lumière de ce qui précède on peut fixer une série de grands objectifs.

1. Il faut définir une métaphore de composantes réutilisables.
2. Il faut définir les mécanismes de leur composition en sous-systèmes et indirectement les mécanismes de l'instanciation de composantes réutilisables.
3. Il faut déterminer la forme que doit prendre la formulation d'un problème de contrôle, entre autres, le formalisme pour l'expression des requis de contrôle.

On espère bien sûr pouvoir éviter l'usage d'AFD pour exprimer les requis du contrôle puisque cette forme se révèle particulièrement ardue à utiliser et cryptique à lire. Il y a là

une difficulté qui mérite d'être soulignée puisque ces automates de spécification (souvent appelés *automates de langage légal*) sont la forme *consacrée* de spécification des requis de contrôle pour SCT. La nécessité d'en changer pour une autre forme de spécification des requis de contrôle limite considérablement les alternatives disponibles et peut exiger d'utiliser un hybride combinant plus d'une des variantes de SCT. On ne peut cependant pas sacrifier la possibilité de synthèse, on y perdrait tous les avantages de SCT.

Certains objectifs sont implicites.

- L'expression des requis de contrôle doit pouvoir se faire de façon modulaire, c'est-à-dire permettre de diviser les requis en petits sous-ensembles qu'il est possible de résoudre indépendamment.
- Le formalisme doit idéalement permettre la traçabilité des requis à leur version informelle.
- Il faut déterminer une dynamique des modèles abstraits pour permettre l'élaboration de schémas de génération de code.

Bien sûr il est nécessaire de tenir à l'esprit pendant l'analyse que l'un des objectifs est la *génération automatique de code* pour l'implémentation des solutions de problèmes de contrôle. Cet objectif peut avoir des ramifications partout dans cette étude. Il peut par exemple requérir des notations spéciales au niveau des éléments de modélisation de problèmes de contrôle ou encore au niveau des mécanismes d'instanciation de composantes. Il est important de le garder à l'esprit pendant toute l'analyse.

En ce qui concerne l'usage des procédures de synthèse de SCT, l'objectif consiste à les utiliser telles qu'elles. Le seul degré de liberté réside dans le choix du type de synthèse :

- une synthèse exhaustive (exploration exhaustive de l'espace des états correspondant à la formulation originale de la théorie par Ramadge et Wonham [45]) ;
- une synthèse modulaire (par exemple le cas de HISC, *Hierarchical Interface-based Supervisory Control* de Leduc, Brandin, Lawford et Wonham [35, 38]) ;
- une synthèse par prédicats (cas des STS de Ma et Wonham [39] utilisant des BDD et souvent qualifiée de méthode *symbolique*).

Ce choix dépend intimement du formalisme de modélisation utilisé. Chaque méthode correspond à une variante de la théorie. Des procédures de synthèse sont disponibles pour chacune de ces alternatives.

Méthodologie

Dans un premier temps il est nécessaire de faire un survol de la littérature scientifique, en commençant par une formulation des résultats fondamentaux de SCT dans son cadre linguistique original. Ensuite, il faut s'attarder à identifier parmi les différentes variantes issues de la théorie, quelles sont celles qui sont les plus susceptibles d'apporter des éléments de solution aux problèmes confrontés. Il y en a principalement trois :

1. le contrôle hiérarchique tel que défini par Wong et Wonham dans [58, 59] et [55] ;
2. le contrôle hiérarchique par interface (HISC) tel que défini par Leduc, Lawford, Brandin et Wonham dans [35, 38] ;
3. le contrôle par STS de Ma et Wonham [39], une variante hiérarchique utilisant l'approche par prédicats.

Chacune de ces variantes de la théorie présente une perspective intéressante et est susceptible de procurer des éléments de solution. Mais aucune d'entre elles ne semble, à elle seule, répondre à tous les objectifs fixés, il faut donc en faire une analyse détaillée.

Dans le but de procurer la *génération automatique du code* des implémentations de solutions aux problèmes de contrôle, il est important d'examiner la documentation de recherche pour identifier les alternatives. Il est donc dans l'ordre d'examiner plus en détail comment la théorie elle-même entrevoit la réalisation de superviseurs issus de ses propres procédures de synthèse. Une certaine attention doit donc être consacrée à comprendre les deux réalisations dites *standards* d'un superviseur SCT :

- réalisation par reproduction de la structure de transition du superviseur (l'AFD du système sous contrôle) ;
- réalisation par implémentation d'une loi de contrôle définie sur l'espace des états du système (appelée une SFBC, pour *State Feed-Back Control*).

La réalisation par SFBC revêt un intérêt particulier pour les applications sur PLC, car elle permet une implémentation presque directe de la loi de contrôle sans avoir recours à une structure de transition (souvent trop volumineuse pour contempler une implémentation sur PLC). Il existe une reformulation de SCT à base de prédicats dont le résultat de synthèse est précisément une SFBC. Cette reformulation s'applique assez généralement à d'autres variantes de la théorie et il est utile d'en donner les fondements.

Une fois la revue de littérature terminée, il faut s'attarder à définir ce que sont les composantes réutilisables. Les composantes doivent répondre à plusieurs critères. On doit définir un mécanisme pour leur abstraction, c'est-à-dire une forme d'encapsulation. Ce mécanisme doit être en accord avec la théorie. Dans ce cas il s'agit de fournir un mécanisme qui permet une encapsulation de la dynamique de la composante sans compromettre la possibilité de synthèse. Un examen attentif est ici nécessaire et certains résultats théoriques peuvent avoir à être développés. On se propose de se livrer à une étude critique des meilleurs candidats, parmi les variantes de SCT, à l'élaboration d'une métaphore de composantes réutilisables, pour identifier les éléments que chacune d'entre elles peut procurer dans ce but. À cette fin il faut développer un modèle préliminaire de composante puis identifier les critères de cette analyse critique. Sur la base des résultats de l'analyse critique, on se propose d'établir le modèle détaillé de la métaphore de composantes réutilisables proposée.

La définition de la métaphore de composantes réutilisables doit être assortie de notations appropriées pour exprimer la structure de ces composantes. Une composante n'est en fait que l'*abstraction* d'un sous-système sur lequel on a défini et résolu un problème de contrôle dont on veut abstraire la solution pour en encapsuler les requis de contrôle. Par ricochet, une composante réutilisable est donc la solution à un problème de contrôle complet. Aussi est-il nécessaire de commencer à élucider la façon dont les composantes sont liées entre elles par les requis de contrôle. Il faut déterminer un formalisme d'expression des requis de contrôle et la méthode de composition des composantes réutilisables en sous-systèmes. Ce qui soulève la question des mécanismes de l'instanciation qu'il faut, à ce point, définir et la question de l'opérateur de composition. L'examen de cas types peut servir de guide dans la détermination de ces éléments. Il est ainsi utile de définir un catalogue de composantes dans le but de les utiliser dans une étude de cas.

Finalement il faut examiner la production de code. Pour la génération de code exécutable sur PLC, la reproduction d'une structure de transition semble clairement à proscrire car elle se révèle généralement trop volumineuse (voir [7]). Il faut donc procéder à une implémentation par SFBC. Les composantes doivent offrir un modèle de leur dynamique interne pour les lier à la génération de code. Il faut définir les éléments de modélisation nécessaires et les notations correspondantes.

On se propose de valider l'ensemble des résultats obtenus par une étude de cas d'une envergure réaliste sur l'usine-école *MPS* du laboratoire des systèmes réactifs.

Contributions et résultats

En guise de contribution, cette thèse précise les éléments suivants.

- Un modèle formel pour supporter la notion de composante réutilisable basé sur une variante hiérarchique de SCT. Ce modèle s'articule autour de trois notions : implémentation, contrôle et interface. Ce modèle a pour but de procurer autant la réutilisation des spécifications de contrôle (notions d'implémentation et de contrôle local) que la réutilisation pure et simple de stéréotypes de comportement par encapsulation et abstraction (notion d'interface).
- Associée au modèle formel, on y développe une notion précise d'instanciation de composantes à travers une adaptation de leur interface.
- Les propriétés que doivent remplir les composantes pour garantir un fonctionnement contrôlable et non bloquant, et donc permettre l'usage des procédures de synthèse de SCT, sont précisées.
- Différents opérateurs de composition en sous-systèmes sont considérés pour la composition horizontale (en sous-systèmes), la composition verticale (réabstraction) et la superposition d'un contrôle local.
- Pour garantir que les propriétés des composantes sont préservées par les trois formes de composition, une règle simple pour l'abstraction de modèle est formulée. Le but visé est de permettre l'agrégation pyramidale d'un nombre arbitraire d'instances de composantes sur un nombre arbitraire de niveaux d'agrégation.

D'un côté plus pragmatique, cette thèse s'attache aussi à développer un certain nombre d'éléments semi-formels pour procurer la génération automatique du code de l'implémentation.

- En plus du model formel, on y définit un diagramme d'assemblage (pour spécifier la structure des sous-systèmes) et une spécification de contrôle. La spécification de contrôle comprend une définition du problème de contrôle et sa solution obtenue par synthèse. Une attention particulière est ici portée à la notation utilisée pour qu'elle

soit utilisable par des non-spécialistes.

- On y définit un modèle de la dynamique abstraite de fonctionnement d'une agrégation d'instances propre à servir de base à la génération de code et à l'insertion de code d'intégration (*glue code* en anglais).

Le tout est assorti d'une étude de cas d'une envergure et d'une difficulté suffisante pour pouvoir tirer des conclusions utiles sur la métaphore de composantes réutilisables proposée. Le code PLC généré pour l'étude de cas utilise le langage SCL (*Structured Control Language*, défini dans [50]) tel qu'implémenté par ***Siemens***^{TM} pour correspondre à la norme internationale IEC 61131-3 (catégorie *Structured Text*).

La plupart de ces contributions ne sont pas de l'ordre de la recherche fondamentale. Mais un examen exhaustif et approfondi de la théorie s'avère nécessaire pour dégager les principes d'une métaphore de composantes réutilisables SCT viable. Cet examen révèle en effet que la plupart des résultats théoriques nécessaires sont déjà là et ne requièrent que de légères adaptations. Mais ces résultats sont disséminés dans une myriade de documents et utilisent souvent des formulations si différentes qu'il est parfois difficile de les relier entre eux pour former un tout cohérent. C'est là l'originalité de la démarche exposée dans cette thèse. L'assemblage d'un hybride composé de deux variantes hiérarchiques de SCT associés à des concepts en provenance du génie logiciel, pour former une métaphore de composantes réutilisables viable que l'on peut éventuellement intégrer à un environnement de conception assistée.

Dans la documentation de recherche sur SCT, les aspects concernant l'implémentation des résultats de synthèse et la réutilisation au cours de la modélisation sont négligés. Ce sont en effet des problèmes difficiles et peu reliés (du moins en apparence) à l'aspect fondamental de la théorie. Mais on peut relever que la carence en modélisation réduit les chercheurs eux-mêmes à la modélisation *ad hoc*, vulnérable aux erreurs et limitée quant à la taille des modèles. Il en résulte souvent des modèles de qualité douteuse et la recherche se trouve souvent cantonnée à l'examen de problèmes de taille réduite dont le potentiel est modeste. Ainsi même la recherche peut bénéficier de la disponibilité d'une métaphore viable de composantes réutilisables en SCT.

Organisation de la thèse

Le chapitre 1 s'attarde à faire un survol de la littérature scientifique pertinente et à définir les bases théoriques nécessaires à cette étude. Le chapitre 2 documente les variantes de la théorie jugées les plus utiles dans l'optique de cette thèse. Le chapitre 3 définit la métaphore de composantes réutilisables proposée dans un détail suffisant pour se livrer ensuite à une critique des variantes vues au chapitre 2 dans le but d'identifier le support théorique nécessaire à son implémentation. On y trouve aussi quelques résultats complémentaires à la théorie du contrôle hiérarchique de Wong et Wonham [55, 58, 59] nécessaires à la formulation de la métaphore de composantes réutilisables. Le chapitre 4 développe en détail les concepts et les notations nécessaires pour supporter la spécification de problèmes de contrôle et de composantes réutilisables. Il s'attarde particulièrement sur la composition de sous-systèmes à partir de composantes réutilisables, sur l'expression des requis de contrôle ainsi que sur la définition du support nécessaire à la génération de code. Il développe ensuite quelques exemples types de composantes réutilisables. Le chapitre 5 traite en détail d'une étude de cas illustrant l'usage de composantes réutilisables (définies dans un catalogue en annexe D) dans la solution d'un problème de contrôle. La conclusion dresse un inventaire des principales contributions, procède à une critique du travail accompli et indique certaines directions de recherche potentielles.

Théorie du contrôle supervisé

Chapitre 1

Rappels sur la théorie du contrôle supervisé

L'essentiel du contenu de ce chapitre provient de la littérature désormais classique sur la théorie du contrôle supervisé habituellement désignée par son acronyme SCT (de l'anglais *Supervisory Control Theory*). En plus des articles fondamentaux sur le sujet dus à Ramadge et Wonham [45, 43, 44], le lecteur peut se reporter avec avantage aux notes de Wonham [62] pour un traitement plus complet, et des commentaires souvent révélateurs. Un traitement alternatif, plus orienté sur la notion de *générateur* de langages ainsi que sur les notions de la théorie des treillis nécessaires à l'établissement des résultats fondamentaux de la théorie, est donné par Kumar et Garg [32]. Le lecteur francophone peut se référer à la thèse de Gaudin [23] dont la section sur les rappels théoriques est particulièrement claire.

1.1 Préliminaires

SCT utilise des concepts de la théorie des langages et de la théorie des automates. Dans ce contexte formel, elle développe certaines notions qui lui sont propres telles que le modèle linguistique d'un *système à événements discrets* (SED), et la notion conséquente de générateur de langages. Il importe de bien situer ces concepts de base qui sont implicites au reste de la théorie.

1.1.1 Systèmes à événements discrets et langages

Les SED sont caractérisés par un espace d'états discrets et une dynamique d'évolution par événements. Lorsqu'un événement se produit le système change d'état ; l'occurrence d'un événement est instantanée. L'ensemble des événements distincts se produisant au sein d'un système est généralement considéré fini et peut être représenté par un alphabet de symboles Σ. L'ensemble de toutes les séquences d'événements (les traces) que le système peut générer pendant son fonctionnement peut être assimilé à un langage $H \subseteq \Sigma^*$, appelé langage *généré* du SED.

Dans ce contexte, un langage L est simplement défini comme un sous-ensemble de Σ^*, l'ensemble de toutes les chaînes de longueur finie que l'on peut former à partir de symboles appartenant à Σ. La chaîne ne contenant aucun symbole, dénotée par le symbole ε, fait aussi partie de cet ensemble, c'est-à-dire $\varepsilon \in \Sigma^*$.

Parmi les séquences d'événements qu'un SED peut générer, certaines ont habituellement un statut spécial puisqu'elles décrivent le déroulement de tâches complètes au sein du système. Ces dernières constituent elles-mêmes un sous-langage $H_m \subseteq H$ du langage généré, appelé langage marqué du SED. La paire de langages (H, H_m) constitue un modèle linguistique du SED en ce sens qu'elle en caractérise complètement le comportement.

Dans le comportement généré d'un SED, il est possible éventuellement de trouver des chaînes $s \in H$ qui ne peuvent jamais être complétées pour former une tâche complète. Le système comporte alors une ou plusieurs impasses (statiques ou dynamiques) ; on dit qu'il est *bloquant*. Il est aussi utile de définir ce qu'est un système non bloquant en termes de son modèle linguistique (H, H_m). À cette fin il faut examiner la notion de langage préfixe clos. Intuitivement, un langage $L \subseteq \Sigma^*$ est préfixe clos si, pour toute chaîne $s \in L$, l'ensemble des préfixes de s fait aussi partie du langage.

Pour $s, u, v \in \Sigma^*$, on dit que u est un préfixe de s, noté $u \leq s$, si $s = uv$. Par définition s et ε sont des préfixes de s puisque $s\varepsilon = \varepsilon s = s$. La fermeture préfixée d'un langage $L \subseteq \Sigma^*$ se définit alors comme suit :

$$\overline{L} := \{u \in \Sigma^* \mid (\exists s \mid s \in L : u \leq s)\}.$$

Lorsque $L = \overline{L}$ on dit que L est préfixe clos.

Par définition d'un modèle linguistique (H, H_m), H est préfixe clos car il contient toutes les chaînes d'événements que le système peut générer et donc, nécessairement, tous leurs préfixes. De plus, pour que le système soit non bloquant, il faut que toute chaîne $s \in H$ puisse éventuellement être complétée en une chaîne appartenant à H_m. En d'autres termes, toute chaîne $s \in H$ doit être un préfixe d'une chaîne de H_m, ce qui revient à dire que $H \subseteq \overline{H}_m$. Mais puisque $H_m \subseteq H$ par définition, la fermeture préfixée étant monotone par rapport à l'inclusion d'ensembles, alors $\overline{H}_m \subseteq \overline{H} = H$, et donc $H = \overline{H}_m$.

Ainsi un SED dont le modèle est donné par la paire de langages (H, H_m) est *non bloquant* si $H = \overline{H}_m$.

1.1.2 Composition de langages

La *composition synchrone* est l'opération principale sur les langages en SCT. Elle sert à décrire le comportement d'un SED complexe à partir des comportements de SED plus simples (composition modulaire des comportements), ou à spécifier des sous-langages du comportement d'un SED ; par exemple la spécification d'un sous-langage contrôlable ou la spécification d'un processus devant être implémenté.

Pour $\Sigma_i \subseteq \Sigma$ (i fini arbitraire), la projection naturelle $P_{\Sigma_i} : \Sigma \to \Sigma_i$ est définie par

$$\begin{aligned} P_{\Sigma_i}(\varepsilon) &:= \varepsilon \\ P_{\Sigma_i}(\sigma) &:= \begin{cases} \varepsilon & \text{si } \sigma \notin \Sigma_i \\ \sigma & \text{si } \sigma \in \Sigma_i \end{cases} \\ P_{\Sigma_i}(s\sigma) &:= P_{\Sigma_i}(s)P_{\Sigma_i}(\sigma) \quad \text{pour } s \in \Sigma^* \text{ et } \sigma \in \Sigma. \end{aligned}$$

Pour $s \in \Sigma^*$, $P_{\Sigma_i}(s)$ efface de s tout $\sigma \in \Sigma \setminus \Sigma_i$ en respectant l'ordre des éléments de la chaîne originale. Ce qui permet de définir la préimage dans Σ^* d'un langage $L \subseteq \Sigma_i^*$, $P_{\Sigma_i}^{-1} : 2^{\Sigma_i^*} \to 2^{\Sigma^*}$ par

$$P_{\Sigma_i}^{-1}(L) := \{s \in \Sigma^* \mid P_{\Sigma_i}(s) \in L\}.$$

L'effet de $P_{\Sigma_i}^{-1}$ sur une chaîne $s \in L$ consiste à intercaler dans cette dernière (de toutes les façons possibles) toutes les chaînes de $(\Sigma \setminus \Sigma_i)^*$ sans déranger l'ordre relatif des éléments originaux de s.

Finalement pour $L_i \in \Sigma_i^*$ $(i = 1, 2)$ et $\Sigma = \Sigma_1 \cup \Sigma_2$, où il est possible que $\Sigma_1 \cap \Sigma_2$ soit non vide, le produit synchrone $L_1 \parallel L_2 \in \Sigma^*$ est donné par

$$L_1 \parallel L_2 := P_{\Sigma_1}^{-1}(L_1) \cap P_{\Sigma_2}^{-1}(L_2), \tag{1.1}$$

d'où $s \in L_1 \parallel L_2$ si et seulement si $P_{\Sigma_1}(s) \in L_1$ et $P_{\Sigma_2}(s) \in L_2$. L'effet du produit synchrone relativement à deux chaînes $s_1 \in L_1$ et $s_2 \in L_2$ est donc le suivant.

- Lorsque $\Sigma_1 \cap \Sigma_2 = \emptyset$, on intercale n'importe quelle portion de s_1 dans s_2 (et réciproquement) de toutes les façons, en respectant l'ordre relatif des éléments des chaînes originales. Il s'agit d'un cas particulier appelé le *produit intercalé* qui correspond au déroulement asynchrone en parallèle de deux systèmes indépendants.
- Lorsque $\Sigma_1 = \Sigma_2$, si $s_1 = s_2 = s$ alors $s \in L_1 \parallel L_2$ sinon $s \notin L_1 \parallel L_2$. La synchronisation entre les deux systèmes est complète et $L_1 \parallel L_2 = L_1 \cap L_2$. Ce cas particulier est souvent appelé *meet* en anglais.
- Lorsque $\Sigma \neq \Sigma_1 \cap \Sigma_2 \neq \emptyset$, le résultat est un mélange de produit intercalé avec synchronisation sur les éléments communs aux deux langages. Cette situation représente le déroulement de deux systèmes en parallèle devant se synchroniser lorsqu'ils effectuent une transition sur un événement qu'ils ont en commun.

Le produit synchrone se généralise à un nombre fini arbitraire de langages puisqu'il est associatif.

1.1.3 Système à événements discrets comme générateur de langages

Le modèle linguistique d'un SED en caractérise le comportement. Cependant, pour exprimer la mécanique du contrôle au sein d'un SED, il est utile de s'en donner un modèle abstrait.

Un SED *génère* des chaînes d'un langage ce qui suggère un certain arbitraire. Il n'est pas possible de savoir quelle chaîne du langage sera générée lors d'une séquence de marche du système. Le modèle linguistique ne fait que circonscrire l'ensemble des possibilités. Le modèle suggéré par la théorie est calqué sur celui d'un *automate fini déterministe* (AFD), soit

$$\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$$

où

- Q est l'ensemble des états du système (au moins dénombrable sinon fini) ;
- Σ est un ensemble fini de symboles dénotant les événements pouvant se produire au sein du système ;
- $\delta : Q \times \Sigma \rightarrow Q$ est une fonction partielle de transition entre états du système ;
- $q_0 \in Q$ est l'état initial du système ;
- $Q_m \subseteq Q$ est l'ensemble des états marqués du système.

Puisque δ est une fonction partielle, la notation $\delta(q, \sigma)!$ est utilisée pour signifier que $\delta(q, \sigma)$ est définie. Il est aussi utile de définir la fonction partielle $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ comme suit, pour $s \in \Sigma^*$ et $\sigma \in \Sigma$:

$$\begin{aligned} \hat{\delta}(q, \varepsilon) &:= q \\ \hat{\delta}(q, s\sigma) &:= \delta(\hat{\delta}(q, s), \sigma), \text{ pourvu que } \hat{\delta}(q, s)! \text{ et } \delta(\hat{\delta}(q, s), \sigma)!. \end{aligned}$$

Enfin, par abus de notation, on utilise δ en lieu et place de $\hat{\delta}$ lorsque le contexte est clair. Pour fin de contrôle, l'alphabet Σ est partitionné en deux sous-ensembles :

- Σ_c, l'ensemble des événements contrôlables ;
- Σ_u, l'ensemble des événements incontrôlables.

L'inhibition d'événements contrôlables est le seul mécanisme servant à restreindre le comportement du système.

Au sein du système les événements se produisent instantanément et sont visibles à un agent externe, mais selon un mécanisme non spécifié par le modèle. En ce sens le fonctionnement du système semble non déterministe, bien que le modèle utilise une structure de transition déterministe.

L'AFD $\mathbf{G}$ est appelé un générateur modélisant un SED $(L(\mathbf{G}), L_m(\mathbf{G}))$, où $L(\mathbf{G})$ représente le langage généré par le SED défini par

$$\overline{L(\mathbf{G})} = L(\mathbf{G}) := \{s \in \Sigma^* \mid \delta(q_0, s)!\},$$

et $L_m(\mathbf{G})$ représente le langage marqué du SED défini par

$$L_m(\mathbf{G}) := \{s \in \Sigma^* \mid \delta(q_0, s) \in Q_m\}.$$

Peu importe le SED modélisé, $\emptyset \subseteq L_m(\mathbf{G}) \subseteq L(\mathbf{G})$ et en général $\varepsilon \in L(\mathbf{G})$ (à moins que $Q = \emptyset$).

L'ensemble des états accessibles de $\mathbf{G}$ est donné par

$$Q_r := \{q \in Q \mid (\exists s \mid s \in \Sigma^* : \delta(q_0, s) = q)\}.$$

L'ensemble des états co-accessibles est donné par

$$Q_{cr} := \{q \in Q \mid (\exists s \mid s \in \Sigma^* : \delta(q, s) \in Q_m)\}.$$

Le SED $\mathbf{G}$ est dit accessible si $Q_r = Q$, co-accessible si $Q_{cr} = Q$ et *soigné* (en anglais *trim*) si ces deux conditions sont remplies simultanément.

Le SED $\mathbf{G}$ est dit non bloquant si tout état accessible est aussi co-accessible (c'est-à-dire $L(\mathbf{G}) = \overline{L_m(\mathbf{G})}$). En particulier $\mathbf{G}$ est non bloquant s'il est soigné (la réciproque n'est pas nécessairement vraie).

Finalement, pour $K \in \Sigma^*$, un générateur $\mathbf{G}$ est dit représenter K si :

- $\mathbf{G}$ est non bloquant ;
- $L_m(\mathbf{G}) = K$.

Car dans ces circonstances, $L(\mathbf{G}) = \overline{K}$, bien que $\mathbf{G}$ ne soit pas nécessairement accessible (ni même co-accessible d'ailleurs). Cependant, lorsque l'on affirme que $\mathbf{G}$ représente un langage K, on entend généralement que $\mathbf{G}$ est soigné. Étant donné un générateur $\mathbf{G}$ quelconque, il est toujours possible d'en obtenir une version soignée.

1.1.4 Notions relatives aux générateurs finis

Les AFD constituent un modèle de générateurs pour les SED lorsque l'espace des états est fini. Le cas fini (par opposition à dénombrable) présente un intérêt particulier pour l'implémentation de contrôleurs matérialisant un superviseur SCT, puisqu'il permet le calcul hors ligne d'une solution (un genre de compilation).

En plus des opérations usuelles sur les AFD, certaines opérations ont un intérêt particulier au sein de SCT.

Soit $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ un AFD, $A \subseteq \Sigma$, $E \in 2^Q$ et $f_G^A : 2^Q \times 2^Q$ définie par

$$f_G^A(E) := E \cup \{q \in Q \mid (\exists \sigma \mid \sigma \in A : (\exists q' \mid q' \in E : \delta(q', \sigma)! \wedge q = \delta(q', \sigma)))\}.$$

Alors $f_G^A(E)$ donne l'ensemble des états de Q que l'on peut atteindre à partir d'un état de E par zéro ou une transition sur un événement tiré de A. Intuitivement il est clair que f_G^A est monotone (par rapport à $\subseteq$) et croissante. Puisque Q est fini, elle admet un ou plusieurs points fixes ($f_G^A(x) = x, x \in 2^Q$). En fait, puisque $(2^Q, \subseteq)$ est un treillis complet, l'ensemble des points fixes de f_G^A comporte une borne supérieure $f^{*A}_G(E)$, où $f^{*A}_G : 2^Q \times 2^Q$ est la fermeture disjonctive de f_G^A définie par

$$f^{*A}_G(E) := \bigcup_{i \geq 0} f^{iA}_G(E),$$

où $f^{0A}_G(E) := E$ (l'identité) et $f^{i+1A}_G(E) := f_G^A(f^{iA}_G(E))$.

On peut ainsi définir $R_G^A(E) := f^{*A}_G(E)$ le plus grand ensemble des états accessibles dans $\mathbf{G}$ à partir d'un ensemble arbitraire $E \subseteq Q$, par tirage d'événements dans $A \subseteq \Sigma$. Lorsque $A = \Sigma$ on note simplement $R_G(E)$ et ainsi $R_G(\{q_0\}) = Q_r$, l'ensemble des états accessibles dans $\mathbf{G}$, et $\mathbf{G}$ est dit accessible si $R_G(\{q_0\}) = Q$.

En utilisant un raisonnement similaire on peut obtenir CR_G^A à partir d'une fonction $g_G^A : 2^Q \times 2^Q$ définie par

$$g_G^A(E) := E \cup \{q \in Q \mid (\exists \sigma \mid \sigma \in A : \delta(q, \sigma) \in E)\}.$$

$CR_G^A(E)$ donne le plus grand ensemble d'états dans $\mathbf{G}$ qui soient co-accessibles à au moins un des états de E, par tirage d'événements dans A. Lorsque $A = \Sigma$ on note simplement $CR_G(E)$ et ainsi $CR_G(Q_m) = Q_{cr}$, l'ensemble des états co-accessibles dans $\mathbf{G}$, et $\mathbf{G}$ est dit co-accessible lorsque $CR_G(Q_m) = Q$.

Il est clair que dans le domaine fini, R_G et CR_G sont bien définis et calculables.

Un AFD $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ est considéré soigné si $R_G(\{q_0\}) = CR_G(Q_m)$. Il est considéré non bloquant si $R_G(\{q_0\}) \subseteq CR_G(Q_m)$. Il est évident que si $\mathbf{G}$ est soigné, alors il est non bloquant bien qu'il puisse être non bloquant sans être soigné.

Puisque les AFD sont des générateurs finis, ils peuvent représenter des langages $K \subseteq \Sigma^*$

lorsque $L_m(\mathbf{G}) = K$ pourvu que $\mathbf{G}$ soit non bloquant ; il s'agit alors des langages réguliers. À moins d'avis contraire un AFD $\mathbf{G}$ représentant un langage K est aussi considéré soigné.

Comme chez les langages, le produit synchrone d'AFD permet de composer un système complexe à partir de composantes plus simples. Le produit synchrone de deux AFD $\mathbf{G}_i = (Q_i, \Sigma_i, \delta_i, x_{0,i}, Q_{m,i})$ $(i = 1, 2)$ est un AFD $\mathbf{G}_1 \parallel \mathbf{G}_2 = (Q, \Sigma, \delta, x_0, Q_m)$, où

$$\begin{aligned} Q &:= Q_1 \times Q_2 \\ \Sigma &:= \Sigma_1 \cup \Sigma_2 \\ x_0 &:= (x_{0,1}, x_{0,2}) \\ Q_m &:= Q_{m,1} \times Q_{m,2} \end{aligned}$$

et pour $q = (q_1, q_2) \in Q$, $\sigma \in \Sigma$ alors

$$\delta(q, \sigma) := \begin{cases} (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)) & \text{si } \sigma \in \Sigma_1 \cap \Sigma_2,\ \delta_1(q_1, \sigma)!,\ \delta_2(q_2, \sigma)! \\ (\delta_1(q_1, \sigma), q_2) & \text{si } \sigma \in \Sigma_1 \setminus \Sigma_2,\ \delta_1(q_1, \sigma)! \\ (q_1, \delta_2(q_2, \sigma)) & \text{si } \sigma \in \Sigma_2 \setminus \Sigma_1,\ \delta_2(q_2, \sigma)! \\ \textit{indéfinie} & \text{autrement.} \end{cases} \tag{1.2}$$

Le produit synchrone d'AFD est associatif et s'étend naturellement à un nombre fini d'AFD.

Pour un ensemble d'AFD $\mathbf{G}_i$ $(i = 0, 1, \ldots, n)$ la relation entre produit synchrone d'AFD et produit synchrone de langages est la suivante :

$$L\left(\parallel_{i \le n} \mathbf{G}_i\right) = \parallel_{i \le n} L(\mathbf{G}_i)\ . \tag{1.3}$$

1.2 Supervision et contrôle

Soit un SED modélisé par un générateur $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$, dont l'alphabet est partitionné en événements contrôlables Σ_c et événements incontrôlables Σ_u ; $\Sigma = \Sigma_c \cup \Sigma_u$. Le contrôle ne peut s'effectuer que par inhibition d'événements contrôlables. Un mécanisme de contrôle simple consiste à présenter au système à tout moment un ensemble d'événements permis comme configuration de contrôle. L'ensemble des configurations de contrôle

est ainsi défini par

$$\mathcal{C} := \{\Sigma' \subseteq \Sigma \mid \Sigma_u \subseteq \Sigma'\}.$$

Puisque les événements incontrôlables ne peuvent jamais être inhibés, toutes les configurations de contrôle contiennent Σ_u.

On définit une loi de contrôle sur $\mathbf{G}$ comme une fonction

$$V : L(\mathbf{G}) \to \mathcal{C}.$$

On utilise la notation $V/\mathbf{G}$ pour la paire $(\mathbf{G}, V)$ et on parle de $\mathbf{G}$ *sous* V pour signifier le fonctionnement en boucle fermée de $\mathbf{G}$ sous le contrôle de la loi V. On appelle $V/\mathbf{G}$ un système supervisé et V un *superviseur* pour $V/\mathbf{G}$.

Le comportement de $V/\mathbf{G}$ est caractérisé par la paire de langages $(L(V/\mathbf{G}), L_m(V/\mathbf{G}))$, où $L(V/\mathbf{G}) \subseteq L(\mathbf{G})$ est défini par

1. $\varepsilon \in L(V/\mathbf{G})$;
2. si $s \in L(V/\mathbf{G})$, $\sigma \in V(s)$ et $s\sigma \in L(\mathbf{G})$, alors $s\sigma \in L(V/\mathbf{G})$;
3. aucune autre chaîne n'apparaît dans $L(V/\mathbf{G})$.

Clairement, $\{\varepsilon\} \subseteq L(V/\mathbf{G}) \subseteq L(\mathbf{G})$ et $L(V/\mathbf{G})$ est un langage non vide et préfixe clos. Aussi le comportement marqué du système supervisé est défini par

$$L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap L_m(\mathbf{G}),$$

c'est-à-dire que les comportements terminaux de $V/\mathbf{G}$ sont ceux de $\mathbf{G}$ qui subsistent sous l'action de V. Ainsi, $\emptyset \subseteq L_m(V/\mathbf{G}) \subseteq L_m(\mathbf{G})$, et on dit que V est non bloquante sur $\mathbf{G}$ si

$$\overline{L_m(V/\mathbf{G})} = L(V/\mathbf{G}).$$

1.2.1 Existence de superviseurs

Il est important d'établir quelles propriétés doivent posséder les langages des systèmes supervisés permettant l'usage d'une loi de contrôle telle que définie précédemment.

On dit qu'un langage $K \subseteq \Sigma^*$ est *contrôlable* (implicitement par rapport à $\mathbf{G}$ et Σ_u) si

$$(\forall s, \sigma \mid s \in \overline{K}, \sigma \in \Sigma_u : s\sigma \in L(\mathbf{G}) \Rightarrow s\sigma \in \overline{K}).$$

C'est-à-dire que tout comportement permis dans le système supervisé ($s \in \overline{K}$) pouvant être immédiatement suivi d'un événement incontrôlable dans le système à contrôler ($s\sigma \in L(\mathbf{G})$ pour $\sigma \in \Sigma_u$) doit aussi être permis dans le système supervisé ($s\sigma \in \overline{K}$). En termes clairs, le comportement marqué du système supervisé $L_m(V/\mathbf{G})$ ne doit jamais requérir l'inhibition d'un événement incontrôlable lorsque ce dernier peut se produire dans le système à contrôler. Cette propriété peut s'énoncer de façon plus succincte si on définit que pour $K \subseteq \Sigma^*$, alors $K\Sigma_u$ représente l'ensemble des chaînes $s\sigma$ où $s \in K$ et $\sigma \in \Sigma_u$. Avec cette notation, K est contrôlable si et seulement si

$$\overline{K}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{K}.$$

Clairement, $\emptyset$, $L(\mathbf{G})$ et Σ^* sont des langages contrôlables par rapport à $\mathbf{G}$.

La contrôlabilité est une propriété du langage $\overline{K} \cap L(\mathbf{G})$ et non de K, ce qui est cohérent avec la loi de contrôle V qui n'est définie que sur $L(\mathbf{G})$. En effet pour une chaîne $s \in K \setminus L(\mathbf{G})$ théoriquement admissible, $s\sigma \notin L(\mathbf{G})$ ne peut jamais se produire dans $\mathbf{G}$ (quel que soit σ). Il est alors immatériel que $s\sigma$ appartienne ou non à $\overline{K}$ puisqu'aucune action de contrôle n'est requise.

Le lien entre la propriété de contrôlabilité et le contrôle exercé par une loi de contrôle telle que V est clair. Mais V doit cependant aussi permettre un contrôle non bloquant du système.

Un langage K est dit *marqué* par un langage L (ou L-marqué) si tous les préfixes de K appartenant à L appartiennent aussi à K. Succinctement pour $K, L \subseteq \Sigma^*$, si $\overline{K} \cap L \subseteq K$ alors K est L-marqué. Si en plus $K \subseteq L$ alors $K = \overline{K} \cap L$ et K est alors dit fermé sur L ou L-fermé.

La L-fermeture (ou plus généralement le L-marquage) de K par L assure que toutes les chaînes terminales dans L (c'est-à-dire marquées) qui sont un préfixe d'une chaîne dans K sont aussi terminales dans K. Autrement dit, dans le cas où K est L-fermé ($K \subseteq L$), L détermine le *marquage* de K, et on peut dire que K est *non marquant*. Dans le cas où

K est arbitraire et L-marqué la situation est la même dans $K \cap L$ bien que K puisse être *marquant* dans $K \setminus L$.

Le lien entre la L-fermeture de $K \subseteq L$ et V devient évident lorsque l'on considère $L = L_m(\mathbf{G})$, pour un générateur $\mathbf{G}$ représentant L, et $K = L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap L_m(\mathbf{G})$, le comportement du système supervisé tel que défini pour V.

Théorème 3.4.1. *[[62], section 3.4] Soit un générateur* $\mathbf{G}$ *et* $\emptyset \neq K \subseteq L_m(\mathbf{G})$. *Il existe une loi de contrôle non bloquante* V *sur* $\mathbf{G}$ *telle que* $L_m(V/\mathbf{G}) = K$ *si et seulement si*

i) K *est contrôlable (par rapport à* $\mathbf{G}$*) ;*

ii) K *est* $L_m(\mathbf{G})$*-fermé.*

Corollaire au théorème 3.4.1. *[[62], section 3.4] Soit un générateur* $\mathbf{G}$ *et* $\emptyset \neq K \subseteq L(\mathbf{G})$, K *préfixe clos. Il existe une loi de contrôle* V *sur* $\mathbf{G}$ *telle que* $L(V/\mathbf{G}) = K$ *si et seulement si* K *est contrôlable (par rapport à* $\mathbf{G}$*).*

Dans ce qui précède, la détermination des chaînes terminales du système (les tâches complètes) est la prérogative exclusive du générateur $\mathbf{G}$. En particulier ceci veut dire que si la chaîne $s \in \overline{K}$ se trouve à être une tâche complète dans $\mathbf{G}$ ($s \in L_m(\mathbf{G})$), elle doit nécessairement être aussi marquée dans K et ce même si elle ne représente pas une tâche complète du processus représenté par le langage K. C'est ce qui contraint à exiger la propriété de $L_m(\mathbf{G})$-fermeture dans le théorème 3.4.1. Ce genre de supervision est dite *non marquante* et pour la désigner on utilise l'acronyme NSC de l'anglais *Nonblocking Supervisory Control*. Par abus de langage on parle aussi d'une NSC V (par référence à la loi de contrôle). Il est cependant possible (et peut-être plus naturel) de définir une loi de contrôle *marquante* (MNSC – *Marking, Nonblocking Supervisory Control*) où la loi de contrôle est couplée à un dispositif (non spécifié) permettant de détecter les chaînes terminales du processus représenté par le langage K et de les marquer.

On définit une MNSC V (c'est-à-dire une loi de contrôle marquante) sur un générateur $\mathbf{G}$ comme une fonction $V : L(\mathbf{G}) \rightarrow C$ comme précédemment, à l'exception du fait que sous une telle loi, le comportement marqué de la boucle de rétroaction est donné par $L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap M$ pour $M \subseteq L_m(\mathbf{G})$.

Théorème 3.4.2. *[[62], section 3.4] Soit* G *un générateur et* $\emptyset \neq K \subseteq L_m(\mathbf{G})$. *Il existe une MNSC* V *sur* G *telle* $L_m(V/\mathbf{G}) = K$ *si et seulement si* K *est contrôlable par rapport à* G.

Dans tout ce qui précède, la loi de contrôle peut être définie, pour tout $s \in \overline{K}$, de la façon suivante :

$$V(s) := \Sigma_u \cup \left\{\sigma \in \Sigma_c \mid s\sigma \in \overline{K}\right\}.$$

1.2.2 Sous-langages contrôlables et contrôle optimal

Dans ce qui précède, le générateur G modélise la machine sur laquelle un *processus*, spécifié par un langage K, doit être implémenté. Bien sûr G et K partagent le même alphabet $\Sigma = \Sigma_c \cup \Sigma_u$, où $\Sigma_c \cap \Sigma_u = \emptyset$. Les résultats énoncés jusqu'à maintenant permettent d'affirmer que si K présente certaines propriétés (contrôlabilité et potentiellement fermeture relative par rapport à G) alors G peut être contrôlé par le biais d'une loi de contrôle V de sorte à produire le comportement décrit par K, le processus. Il est évident que trouver un tel K n'est pas trivial.

En général la situation se présente différemment. On élabore d'abord la *spécification* d'un processus sous la forme d'un langage $E \subseteq \Sigma^*$ qui constitue un *objectif de contrôle* pour le générateur G. Ce langage est rarement contrôlable ou relativement fermé a priori. Les résultats principaux de la théorie permettent d'affirmer qu'il existe des sous-langages de E possédant les propriétés requises (contrôlabilité et si nécessaire fermeture relative) pour l'implémentation par une loi de contrôle.

Les principaux résultats de SCT sont basés sur l'observation que pour tout alphabet Σ fini, $(2^{\Sigma^*}, \subseteq)$ forme un treillis complet. Les arguments de la théorie sont des arguments de la théorie des treillis dont on peut trouver une discussion détaillée dans [32]. L'ensemble des résultats qui suivent sont directement levés de [62] (au chapitre 3), où on trouve aussi les démonstrations.

Dans cette section, on suppose un générateur G d'alphabet $\Sigma = \Sigma_c \cup \Sigma_u$, où $\Sigma_c \cap \Sigma_u = \emptyset$, et un objectif de contrôle $E \subseteq \Sigma^*$ arbitraire. On définit $C(E) \subseteq 2^E$, l'ensemble des

sous-langages contrôlables de E, par

$$\mathcal{C}(E) := \left\{K \subseteq E \mid \overline{K}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{K}\right\},$$

et $\mathcal{F}(E) \subseteq 2^E$, l'ensemble des sous-langages L-fermés de E ($L \subseteq \Sigma^*$ arbitraire), par

$$\mathcal{F}(E) := \left\{F \subseteq E \mid F = \overline{F} \cap L\right\}.$$

Proposition 3.5.1. *[[62], section 3.5]* $\mathcal{C}(E) \neq \emptyset$ *puisqu'il contient au moins* $K_\emptyset = \{\}$*, le langage vide, et* $\mathcal{C}(E)$ *est fermé sous l'union de langages. En particulier,* $\mathcal{C}(E)$ *admet un borne supérieure dénotée* $\sup\mathcal{C}(E)$*, où*

$$\sup\mathcal{C}(E) := \bigcup_{K \in \mathcal{C}(E)} K.$$

$\mathcal{C}(E)$ ne forme pas un sous-treillis complet de 2^E puisqu'il n'est pas fermé sur l'intersection de langages. En revanche, l'ensemble des sous-langages contrôlables et préfixe clos de E, $\left\{K \subseteq E \mid K \in \mathcal{C}(E) \wedge K = \overline{K}\right\}$, forme un sous-treillis complet de 2^E ([62], proposition 3.5.2) et admet une paire de bornes supérieure et inférieure. Aussi $\mathcal{F}(E) \neq \emptyset$ ($K_\emptyset \in \mathcal{F}(E)$) et $\mathcal{F}(E)$ forme un sous-treillis complet de 2^E ([62], proposition 3.5.3).

La proposition suivante est un résultat important qui permet une simplification dans la recherche (ou le calcul) d'une solution à un objectif de contrôle E. Pour mémoire, si $E \supseteq \overline{E} \cap L$, on dit que E est L-marqué.

Proposition 3.5.4. *[[62], section 3.5] Si* $E \subseteq \Sigma^*$ *est* $L_m(\mathbf{G})$*-marqué, alors* $\sup\mathcal{C}(E \cap L_m(\mathbf{G}))$ *est* $L_m(\mathbf{G})$*-fermé.*

Cette dernière proposition permet d'éviter d'avoir à considérer un ensemble $\mathcal{CF}(E)$ des langages contrôlables et relativement fermés pour arriver à une solution pour E. De plus il est assez naturel de considérer lors de l'élaboration d'un objectif de contrôle E, que les comportements terminaux de E correspondent normalement à des tâches complètes dans $\mathbf{G}$. Aussi, si l'agent marquant du système devait absolument être le générateur, l'objectif de contrôle à considérer peut être remplacé par $E' = \overline{E}$, puisque $\overline{E} \cap L_m(\mathbf{G})$ est $L_m(\mathbf{G})$-fermé donc automatiquement $L_m(\mathbf{G})$-marqué.

Théorème 3.5.1. *[[62], section 3.5] Soit $E \subseteq \Sigma^*$, $L_m(\mathbf{G})$-marqué et $K = \sup C(E \cap L_m(\mathbf{G}))$. Si $K \neq \emptyset$, alors il existe une NSC V (non marquante et non bloquante) sur $\mathbf{G}$ telle que $L_m(V/\mathbf{G}) = K$.*

Si K n'est pas vide, alors K représente la solution la moins restrictive permettant le contrôle de $\mathbf{G}$ de sorte que le comportement résultant appartienne à la spécification E sans provoquer d'impasse dans le système. En ce sens, le contrôle exercé par V est dit optimal.

Bien sûr si l'agent de marquage du système peut être incorporé à la loi de contrôle (ou à tout autre agent que le générateur $\mathbf{G}$), alors on obtient un résultat plus simple.

Théorème 3.5.2. *[[62], section 3.5] Soit $E \subseteq \Sigma^*$ et $\emptyset \neq K = \sup C(E \cap L_m(\mathbf{G}))$. Alors il existe une MNSC V (marquante et non bloquante) sur $\mathbf{G}$ telle que $L_m(V/\mathbf{G}) = K$.*

Ce dernier théorème correspond à la situation la plus réaliste où les tâches complètes du processus sont déterminées par l'objectif de contrôle E (la spécification du processus), mais sont constituées d'ensemble de sous-tâches complètes de la machine ($L_m(\mathbf{G})$).

1.3 Synthèse de superviseurs et contrôleurs

Comme le font remarquer Kumar et Garg dans [32], il serait difficile d'appliquer les résultats de SCT en dehors du domaine fini, puisque la vérification des propriétés de contrôlabilité et de fermeture relative est considérée comme indécidable dans le domaine des langages récursivement énumérables. Cependant la grande majorité des systèmes modernes que l'on peut décrire à l'aide de SED admettent une expression finie sous forme d'AFD. Il y a donc avantage à transporter les résultats de la théorie dans le domaine fini des langages réguliers.

1.3.1 Superviseurs adéquats

La notion d'un générateur $\mathbf{S}_{gen}$ agissant comme *superviseur* d'un système contrôlé $V/\mathbf{G}$ permet une redéfinition de V (la loi de contrôle) dans l'espace des AFD, procurant ainsi une base finie pour la spécification de procédures de synthèse.

Pour une loi de contrôle (marquante et non bloquante) V sur un générateur $\mathbf{G}$ telle que $L_m(V/\mathbf{G}) = K$ et $L(V/\mathbf{G}) = \overline{K}$, on dit qu'un générateur $\mathbf{S}_{gen}$ implémente V lorsque

$$L_m(\mathbf{S}_{gen}) \cap L_m(\mathbf{G}) = K \text{ et } L(\mathbf{S}_{gen}) \cap L(\mathbf{G}) = \overline{K}.$$

Or d'après le théorème 3.5.2, $K = \sup\mathcal{C}(E \cap L_m(\mathbf{G}))$ $(E \subseteq \Sigma^*)$ admet une MNSC V sur $\mathbf{G}$. Ainsi, un générateur $\mathbf{K}_{gen}$ représentant K (le comportement en boucle fermée $V/\mathbf{G}$) implémente V. Cependant $\mathbf{K}_{gen}$ n'est pas le seul générateur qui puisse implémenter V et en général, il n'est pas nécessaire qu'un générateur représente le comportement en boucle fermée du système contrôlé, $L_m(V/\mathbf{G})$, pour implémenter V.

De façon générale, pour $S \subseteq \Sigma^*$, si S est contrôlable par rapport à $\mathbf{G}$ et tel que $\overline{S \cap L_m(\mathbf{G})} = \overline{S} \cap L(\mathbf{G})$, alors $K = S \cap L_m(\mathbf{G})$ est contrôlable par rapport à $\mathbf{G}$ et admet une MNSC V telle que $L_m(V/\mathbf{G}) = K$. Ainsi pour un générateur $\mathbf{S}_{gen}$ représentant S,

- $L_m(\mathbf{S}_{gen}) \cap L_m(\mathbf{G}) = S \cap L_m(\mathbf{G}) = K$,
- $L(\mathbf{S}_{gen}) \cap L(\mathbf{G}) = \overline{S} \cap L(\mathbf{G}) = \overline{K}$,

et $\mathbf{S}_{gen}$ implémente V. Lorsque ces conditions sont remplies, on dit que le générateur $\mathbf{S}_{gen}$ est un *superviseur* pour $\mathbf{G}$ (incluant le cas où $L_m(\mathbf{S}_{gen}) \cap L_m(\mathbf{G}) = \emptyset$).

Si l'on admet qu'un générateur $\mathbf{S}_{gen}$ soit dit contrôlable lorsqu'il représente un langage S lui-même contrôlable par rapport à $\mathbf{G}$, on peut alors donner une définition de *superviseur*. Un générateur $\mathbf{S}_{gen}$ est appelé un *superviseur adéquat* pour G lorsque :

i) $\mathbf{S}_{gen}$ est soigné ;

ii) $\mathbf{S}_{gen}$ est contrôlable par rapport à G ;

iii) $\overline{L_m(\mathbf{S}_{gen}) \cap L_m(\mathbf{G})} = L(\mathbf{S}_{gen}) \cap L(\mathbf{G})$.

Bien que (i) ne soit pas nécessaire pour que $\mathbf{S}_{gen}$ implémente V de façon non bloquante, le terme *superviseur* utilisé dans la théorie correspond généralement à la définition de superviseur adéquat.

La propriété (iii), $\overline{L_m(\mathbf{S}_{gen}) \cap L_m(\mathbf{G})} = L(\mathbf{S}_{gen}) \cap L(\mathbf{G})$, n'est pas triviale et peut requérir une vérification à moins d'assurer d'une façon ou d'une autre que $\mathbf{S}_{gen}$ est le générateur non bloquant d'un sous-langage contrôlable de $L_m(\mathbf{G})$. Lorsque $L_m(\mathbf{S}_{gen}) \subseteq L_m(\mathbf{G})$, cette propriété est trivialement vérifiée. Par exemple, pour un SED $\mathbf{G}$ défini sur

un alphabet Σ et un objectif de contrôle $E \subseteq \Sigma^*$, si l'on dispose d'un générateur $\mathbf{E}_{gen}$ représentant E, alors le générateur $\mathbf{K}_{gen}$ représentant $K = \sup\mathcal{C}(L_m(\mathbf{E}_{gen}) \cap L_m(\mathbf{G}))$ est un superviseur adéquat pour $\mathbf{G}$.

Si $\mathbf{S}_{gen} = (X, \Sigma, \xi, x_0, X_m)$ implémente V sur $\mathbf{G}$, $L_m(V/\mathbf{G}) = L_m(\mathbf{S}_{gen}) \cap L_m(\mathbf{G})$ et $L(V/\mathbf{G}) = L(\mathbf{S}_{gen}) \cap L(\mathbf{G})$. Il est ainsi possible de formuler $\psi : X \rightarrow \mathcal{C}$, une loi de contrôle où $\psi(x) := \{\sigma \in \Sigma \mid \xi(x, \sigma)!\}$, dont la dynamique est illustrée en figure 1.1. L'action du contrôle est associée à $\mathbf{S}_{gen}$ qui suit la progression de $\mathbf{G}$ et inhibe $\sigma \notin \psi(x)$ dans $\mathbf{G}$ tant qu'il réside à l'état x. Cette forme de la loi de contrôle est plus exploitable pour l'implémentation de contrôleurs que $V : L(\mathbf{G}) \rightarrow \mathcal{C}$. Le résultat est analogue à un produit synchrone de générateurs ($\mathbf{S}_{gen}$ et $\mathbf{G}$) définis sur le même alphabet Σ,

$$L_m(V/\mathbf{G}) = L_m(\mathbf{S}_{gen}) \parallel L_m(\mathbf{G}) = L_m(\mathbf{S}_{gen} \parallel \mathbf{G}).$$

Si $\mathbf{S}_{gen}$ admet une expression finie (comme dans le cas où $\mathbf{S}_{gen}$ est un AFD), il est possible d'implémenter un contrôleur pour $\mathbf{G}$ basé sur la structure de transition de $\mathbf{S}_{gen}$. On peut donc définir informellement un contrôleur SCT comme l'implémentation matérielle d'un superviseur adéquat $\mathbf{S}_{gen}$ sur une machine $\mathbf{G}$.

1.3.2 Synthèse de superviseurs adéquats

Le problème de synthèse de superviseurs se présente généralement sous la forme suivante :

1. On dispose d'un système pour lequel on élabore un modèle d'AFD, noté $\mathbf{G}$, le générateur du SED. Ce modèle est souvent appelé *modèle du comportement libre du système.*
2. À partir d'une spécification pour un processus que l'on veut implémenter sur $\mathbf{G}$, on élabore un modèle du processus sous forme d'un AFD, noté $\mathbf{E}_{gen}$, qui représente le *langage légal* du processus. Il est habituellement assez naturel de spécifier comme séquences terminales de $\mathbf{E}_{gen}$ des séquences également terminales dans $\mathbf{G}$. La spécification $L(\mathbf{E}_{gen})$ n'est encore qu'un objectif de contrôle.

À partir de ces deux générateurs on cherche un superviseur adéquat $\mathbf{K}_{gen}$ représentant $K = \sup\mathcal{C}(L_m(\mathbf{E}_{gen}) \cap L_m(\mathbf{G}))$. S'il existe, il s'agit de la meilleure solution contrôlable

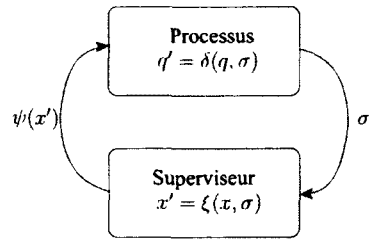


FIGURE 1.1 – Modèle de rétroaction par générateur

au problème d'implémenter $L_m(\mathbf{E}_{gen})$ sur $\mathbf{G}$. Bien sûr, il faut supposer que la spécification $L_m(\mathbf{E}_{gen})$ est adéquate.

Une procédure de synthèse type est donnée par l'algorithme 1.1. Cette procédure calcule l'AFD du superviseur adéquat représentant le comportement du système contrôlé en boucle fermée ($V/\mathbf{G}$). L'algorithme procède au calcul d'un point fixe contrôlable et non bloquant dans l'ensemble des sous-langages de $L_m(\mathbf{E}_{gen}) \cap L_m(\mathbf{G})$ par manipulation de l'automate produit. Le point fixe existe (il peut être vide), chaque itération élague des états, le domaine étant fini l'algorithme termine à coup sûr.

À l'étape 1, l'algorithme élabore d'abord le produit synchrone $\mathbf{S} = \mathbf{G} \parallel \mathbf{E}_{gen}$ formant la base des calculs, puis détermine l'ensemble des états de ce produit qui ne violent pas le critère de contrôlabilité ($good$). Le calcul est arrêté sitôt qu'il est clair que l'état initial viole le critère de contrôlabilité ($x_0 \notin good$), dans ce cas le point fixe est le langage vide.

L'étape 2 travaille sur l'automate produit résiduel duquel tous les états violant le critère de contrôlabilité ont été enlevés. Elle évalue $newbad \subseteq good$, l'ensemble des états bloquants de la solution contrôlable, et $newgood \subseteq good$, l'ensemble des états non bloquants qui demeurent contrôlables malgré le nouvel élagage.

L'algorithme s'arrête à l'étape 3 sitôt que la solution est non vide, contrôlable et non bloquante (c'est-à-dire que l'étape 2 n'a rien changé et que l'AFD résiduel est contrôlable et non bloquant).

Algorithme 1.1: Synthèse de superviseur.

Entrées :

$\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$, un générateur soigné représentant le SED ;
$\mathbf{E}_{gen} = (Y, \Sigma, \gamma, y_0, Y_m)$, un générateur soigné représentant l'objectif de contrôle.

Notes :

$\mathbf{S} = (X, \Sigma, \xi, x_0, X_m)$, *représente l'AFD du produit synchrone ;*
$bad, newbad \subseteq X$ *représentent des ensembles d'états proscrits ;*
$good, newgood \subseteq X$ *représentent des ensembles d'états non proscrits ;*
$\mathbf{S}' = (X', \Sigma, \xi', x_0, X'_m)$, *représente l'AFD résiduel induit par :*
$X' := X \cap good$,
$X'_m := X_m \cap good$
et $\xi' := \xi_{|good}$, *la restriction de* ξ *à* $good \times \Sigma$.

Sorties :

En sortie S' dénote l'AFD (pas nécessairement co-accessible) de la solution si elle existe.

1) Initialisations :
$\mathbf{S} \leftarrow \mathbf{G} \parallel \mathbf{E}_{gen}$
$bad \leftarrow \{x = (q, y) \in X \mid (\exists \sigma \mid \sigma \in \Sigma_u : \delta(q, \sigma)!$ et $\gamma(y, \sigma)$ n'est pas définie$)\}$
$good \leftarrow X \setminus CR_{\mathbf{S}}^{\Sigma_u}(bad)$

2) Itération :
si $x_0 \notin good$ **alors**
$\mathbf{S}' \leftarrow AFD_{\emptyset}$, arrêt, il n'y a pas de solution.
sinon
$\mathbf{S}' \leftarrow \mathbf{S}_{|good}$
$newbad \leftarrow X' \setminus CR_{\mathbf{S}'}(X'_m)$
$newgood \leftarrow X' \setminus CR_{\mathbf{S}'}^{\Sigma_u}(newbad)$

3) Fin :
si $newgood = X'$ **alors**
arrêt, la solution est $trim(\mathbf{S}')$.
sinon
$good \leftarrow newgood$
retourner à l'étape (2)

Chapitre 2

Variantes de la théorie du contrôle supervisé

Kumar et Garg ([32], chapitre 3) donnent un algorithme similaire à l'algorithme 1.1 et montrent que ce genre d'algorithme a une complexité $O(m^2n^2)$ pour G et $\mathbf{E}_{gen}$ comportant respectivement m et n états. Cependant l'usage du produit synchrone pour l'élaboration de générateurs à partir de composantes résulte souvent dans des AFD dont la taille est très élevée, et rend impossible l'application des procédures de synthèse pour des systèmes réalistes. SCT souffre ainsi d'un problème reconnu d'explosion combinatoire de l'espace d'états, et plusieurs variantes de la théorie ont été formulées pour tenter de mitiger ce problème.

2.1 Synthèse modulaire

Lors de la spécification d'un problème de synthèse, il est nécessaire d'élaborer un modèle du comportement libre du système sous forme d'un AFD G (section 1.3.2). La composition synchrone d'AFD fournit une solution simple permettant la modularisation des composantes $\mathbf{G}_i$, $i \in \{1, \ldots, n\}$:

$$\mathbf{G} = \parallel_{i \leq n} \mathbf{G}_i.$$

Il faut aussi élaborer une spécification du processus à implémenter sous forme d'un AFD $\mathbf{E}$ où $E = L(\mathbf{E})$, l'objectif de contrôle du système. Il est assez naturel de penser à diviser ce problème, pour le simplifier, en utilisant la même technique de modularisation utilisée pour obtenir le comportement libre du système. Le but visé est de diviser la spécification E en *modules* E_i, $i \in \{1, \ldots m\}$, donc d'obtenir un ensemble de générateurs $\mathbf{E}_i$ tels que

$$\mathbf{E} = \underset{i \leq m}{\|} \mathbf{E}_i$$

où $E_i = L(\mathbf{E}_i)$.

La modularisation de l'objectif de contrôle soulève naturellement la question de savoir sous quelles conditions l'ensemble des modules de la spécification sont équivalents à un objectif de contrôle spécifié de façon *monolithique* (un seul AFD $\mathbf{E}$).

Deux langages $K_1, K_2 \subseteq \Sigma^*$ sont considérés *modulaires* si $\overline{K_1 \cap K_2} = \overline{K_1} \cap \overline{K_2}$ (on dit aussi que K_1 et K_2 sont *non conflictuels* [62]). Or pour un générateur $\mathbf{G}$ défini sur Σ, et deux langages $K_1, K_2 \subseteq \Sigma^*$ contrôlables par rapport à $\mathbf{G}$ et modulaires entre eux :

$$\begin{aligned}
\overline{(K_1 \cap K_2)}\Sigma_u \cap L(\mathbf{G}) &= (\overline{K_1} \cap \overline{K_2})\Sigma_u \cap L(\mathbf{G}) && K_1, K_2 \text{ modulaires} \\
&= (\overline{K_1}\Sigma_u \cap L(\mathbf{G})) \cap (\overline{K_2}\Sigma_u \cap L(\mathbf{G})) \\
&\subseteq \overline{K_1} \cap \overline{K_2} && K_1, K_2 \text{ contrôlables} \\
&\subseteq \overline{(K_1 \cap K_2)} && K_1, K_2 \text{ modulaires}
\end{aligned}$$

et ainsi $K_1 \cap K_2$ est aussi contrôlable par rapport à $\mathbf{G}$.

Il est facile de démontrer que deux langages préfixe clos définis sur le même alphabet sont toujours modulaires entre-eux. Aussi la définition de contrôlabilité (donnée en section 1.2.1) implique qu'un langage K n'est contrôlable que si et seulement si $\overline{K}$ est contrôlable.

Ainsi pour un ensemble d'objectifs de contrôle $E_i \in \Sigma^*$, $i \in \{1, \ldots, n\}$, l'ensemble des langages $\overline{K_i}$, où $K_i = \sup\mathcal{C}(E_i \cap L_m(\mathbf{G}))$, sont contrôlables et modulaires entre-eux. Par conséquent l'ensemble de générateurs $\mathbf{K}_i$, où $\overline{K_i} = L(\mathbf{K}_i)$, permet d'obtenir un

comportement généré contrôlable par composition synchrone :

$$\begin{aligned} L\left(\parallel_{i\leq n} \mathbf{K}_i\right) &= \parallel_{i\leq n} L(\mathbf{K}_i) \\ &= \parallel_{i\leq n} \overline{K}_i \\ &= \bigcap_{i\leq n} \overline{K}_i. \end{aligned}$$

Mais puisque $\bigcap_{i\leq n} \overline{K}_i \supseteq \overline{\bigcap_{i\leq n} K_i}$, avec une inclusion stricte en général, ce système pourrait s'avérer bloquant, en admettant que $K = \bigcap_{i\leq n} K_i$ soit le langage marqué recherché. Il suffit donc d'assurer que chaque solution partielle soit contrôlable localement pour assurer la contrôlabilité de l'ensemble. Pour cette raison la contrôlabilité est considérée comme une propriété *locale*. Par opposition, l'absence d'impasse requiert de considérer le système dans son ensemble, aussi est-elle considérée comme une propriété *globale* du système.

Pour $E_1, E_2 \subseteq \Sigma^*$, puisque $\sup\mathcal{C}(E_1 \cap E_2)$ est un sous-langage contrôlable de E_1 (et aussi de E_2) dont la borne supérieure est $\sup\mathcal{C}(E_1)$ (respectivement $\sup\mathcal{C}(E_2)$), alors

$$\sup\mathcal{C}(E_1 \cap E_2) \subseteq \sup\mathcal{C}(E_1) \cap \sup\mathcal{C}(E_2)$$

généralement. Aussi, $E_1 \cap E_2 \supseteq \sup\mathcal{C}(E_1) \cap \sup\mathcal{C}(E_2)$ puisque $\sup\mathcal{C}(E_i) \subseteq E_i$ pour $i = 1, 2$. Si $\sup\mathcal{C}(E_i)$ pour $i = 1, 2$ sont modulaires entre-eux, alors $\sup\mathcal{C}(E_1) \cap \sup\mathcal{C}(E_2)$ est un sous-langage contrôlable de $E_1 \cap E_2$ dont la borne supérieure est $\sup\mathcal{C}(E_1 \cap E_2)$, et par conséquent :

$$\sup\mathcal{C}(E_1 \cap E_2) \supseteq \sup\mathcal{C}(E_1) \cap \sup\mathcal{C}(E_2).$$

Ainsi, pour $E_1, E_2 \subseteq \Sigma^*$ tels que $\sup\mathcal{C}(E_1)$ et $\sup\mathcal{C}(E_2)$ sont modulaires entre-eux,

$$\sup\mathcal{C}(E_1 \cap E_2) = \sup\mathcal{C}(E_1) \cap \sup\mathcal{C}(E_2).$$

En synthèse modulaire il faut donc vérifier que l'ensemble des solutions partielles ($K_i = \sup\mathcal{C}(E_i \cap L_m(\mathbf{G}))$) sont modulaires entre elles pour assurer que le système sous contrôle soit non bloquant. Il s'agit d'une forme de symétrie en ce sens qu'aucune des

spécifications partielles ne doit être subordonnée à une autre pour l'accomplissement complet des sous-tâches qu'elle décrit. Pour cette raison, cette forme de modularité est souvent qualifiée de *modularité horizontale* dans la littérature scientifique sur le sujet.

La vérification de modularité entraîne des calculs aussi complexes que ceux d'une synthèse monolithique, elle ne présente donc aucun gain à ce chapitre. Toutefois elle permet une certaine simplification à la démarche d'exprimer l'objectif de contrôle global. Aussi, en plus d'offrir la possibilité de se concentrer sur un ensemble de problèmes simplifiés pour l'élaboration de l'objectif de contrôle (*diviser-pour-régner*), la modularisation horizontale du problème de contrôle ouvre plusieurs possibilités concernant l'*architecture* de la solution.

1. Le contrôleur peut être implémenté de façon *monolithique* ou par un ensemble de *modules* distincts.
2. Dans le cas d'une implémentation modulaire du contrôleur, les modules peuvent être centralisés sur un même processeur ou bien distribués sur plusieurs.

Dans le cas d'une implémentation par modules du contrôleur, il faut bien sûr définir quel est l'effet conjugué des différents modules. La théorie à ce sujet est détaillée dans [62] au chapitre 4. Aussi, si les conditions s'y prêtent, il est possible que l'implémentation de l'ensemble des modules présente une complexité en espace moins grande que celle d'une solution monolithique. Cette question est examinée dans [10] et [11]. Mais en général, la conception d'une spécification modulaire (modularité horizontale) utilise des heuristiques comme celles examinées dans [62] aux sections 4.7 et 4.8. Dans la théorie, de telles heuristiques sont reléguées à la catégorie des *techniques de modélisation* ; on y porte en général peu d'attention.

Bien sûr dans le cas d'une implémentation distribuée de la solution, certains modules pourraient ne pas être en mesure de percevoir tous les événements du système. Ils devraient alors fonctionner dans des conditions dites d'*observation partielle*.

2.2 Contrôle sous observation partielle

Lorsque le contrôleur d'un système ne peut percevoir qu'un sous-ensemble $\Sigma_o \subseteq \Sigma$ des événements se produisant dans le système, la loi de contrôle régissant ce système devra nécessairement prendre la forme d'une fonction dont le domaine est limité à l'ensemble des chaînes d'événements dits observables.

Pour le contrôle sous observation partielle, SCT énonce la propriété d'*observabilité* d'un langage qui permet de démontrer l'existence de contrôleurs non bloquants [32] sous l'hypothèse d'un objectif de contrôle observable, contrôlable et $L_m(\mathrm{G})$-fermé. Cependant l'observabilité n'est pas une propriété stable sous l'union de langages, de sorte qu'il est en général impossible de résoudre le problème de la synthèse non bloquante à partir d'un objectif de contrôle quelconque K.

Pour pallier cette difficulté, SCT élabore la notion de langage normal, plus restrictive que l'observabilité, mais stable sous l'union de langages. Cette propriété permet d'obtenir des contrôleurs non bloquants par synthèse. Comme mentionné dans [62] la principale différence entre un langage normal et un langage observable du point de vue du contrôleur réside dans le fait que pour un langage normal et préfixe clos, la projection $P_{\Sigma_o}(\cdot)$ suffit pour déterminer si une chaîne $s \in \Sigma^*$ est élément de la spécification de l'objectif de contrôle K. Pour un langage observable ce n'est pas le cas en général. Ainsi, pour un langage normal, l'occurrence d'événements non observables, en particulier l'occurrence d'événements contrôlables mais non observables, ne peut pas entraîner le système hors de la spécification. Il n'est donc jamais nécessaire d'inhiber de tels événements. La notion de langage normal est aussi importante parce qu'on a établi que, dans un système où tous les événements contrôlables sont aussi observables ($\Sigma_c \subseteq \Sigma_o$), la propriété de normalité est équivalente à la propriété d'observabilité.

De prime abord, la notion d'événement contrôlable que l'on ne peut pas observer peut sembler bizarre, puisque ces événements correspondent généralement à des commandes devant être émises et que cette fonction incombe en général à la logique de contrôle d'un système. Il faut se rappeler cependant que le modèle théorique de SCT ne formalise pas la notion de commande. Les événements dans un système (contrôlé ou non) s'y produisent spontanément. Le superviseur agit en mode réactif par inhibition d'événements.

2.3 Contrôle hiérarchique

Le contrôle hiérarchique tente d'apporter des solutions aux problèmes de complexité en synthèse de contrôleurs par l'abstraction et la modularité verticale. Par opposition à la modularité horizontale (voir la section 2.1), une structuration arborescente de système donne lieu à une forme de modularité où certains systèmes sont assujettis à d'autres et isolés du système global par une forme d'encapsulation, d'où l'appellation de *modularité verticale.*

Précédemment on a vu que bien que la synthèse modulaire puisse résulter en un ensemble de contrôleurs modulaires dont la complexité globale est moins élevée que pour une solution monolithique, elle n'offre pas vraiment d'avantage concluant quant à la complexité des calculs. Si les contrôleurs résultants doivent être non bloquants globalement, il faut alors vérifier la modularité des $\sup \mathcal{C}(K_i)$, les plus grands sous-langages contrôlables et $L_m(\mathbf{G})$-fermés correspondant aux objectifs de contrôle initiaux. Cette vérification est aussi complexe que la synthèse monolithique, et dans le cas où le résultat n'est pas un ensemble de langages modulaires, aucune heuristique ou algorithme n'existe pour obtenir une solution maximale modulaire.

La théorie du contrôle hiérarchique a été formalisée par Wong dans sa thèse de doctorat [55] et repose sur des travaux antécédents par Zhong et Wonham [66] dont on trouve un bon résumé dans [62]. Ce qui suit est un sommaire tiré de la thèse de Wong ainsi que de plusieurs articles pour la plupart dus à Wong et Wonham ([58, 59, 60]).

2.3.1 Préliminaires

La théorie du contrôle hiérarchique repose sur une reformulation de SCT utilisant des concepts légèrement différents. En plus des notions sur les langages vues au chapitre 1, il est utile d'ajouter quelques autres éléments.

2.3.1.1 Notions algébriques

On se rappelle que pour un alphabet Σ, l'ensemble Σ^* est constitué de toutes les chaînes finies de symboles dans Σ incluant la chaîne vide dénotée par ε. Alors un sous-ensemble

$H \subseteq \Sigma^*$ est appelé un langage sur Σ. Pour $s, u \in \Sigma^*$, la notation $s \leq u$ exprime que s est un préfixe de u. On définit $\overline{H} := \{s \in \Sigma^* \mid (\exists u \mid u \in H : s \leq u)\}$ pour $H \subseteq \Sigma^*$, la fermeture préfixée de H, et on dit que H est *préfixe clos* (ou simplement *fermé*) si $H = \overline{H}$. On dit que pour $L \subseteq \Sigma^*$ fermé (c'est-à-dire $L = \overline{L}$), $\mathcal{P}(L)$ dénote 2^L, l'ensemble des sous-langages de L, et $\mathcal{P}^2(L)$ dénote ainsi l'ensemble des familles de sous-langages de L. Pour $L \subseteq \Sigma^*$ fermé, $\mathcal{F}_L := \{H \subseteq L \mid H = \overline{H}\}$ dénote la famille des sous-langages fermés de L. Pour $L \subseteq \Sigma^*$ fermé et $H \subseteq L$, on définit $pos_L(H) := \{s \in L \mid (\exists u \mid u \in H : u \leq s)\}$, la fermeture postfixée de H dans L. Ainsi $pos_L(H)$ contient toutes les chaînes de L qui constituent une extension d'au moins une chaîne de H. Pour $L \subseteq \Sigma^*$ fermé, on définit

$$Elig_L : L \to \mathcal{P}(\Sigma) : s \longmapsto \{\sigma \in \Sigma \mid s\sigma \in L\}$$

donnant l'ensemble des événements pouvant être générés immédiatement à la suite de s dans L.

Pour un ensemble X quelconque, $\mathcal{E}(X)$ dénote l'ensemble des relations d'équivalence que l'on peut dresser sur X. Une relation d'équivalence $\omega \in \mathcal{E}(X)$ induit une partition de X dont les éléments $[x]_\omega := \{x' \in X \mid x' R_\omega x\}$ sont appelés les *classes d'équivalence* de ω dans X. On parle alors de $X/\omega := \{[x]_\omega \mid x \in X\}$, le quotient de X par ω, et on définit la fonction $p_\omega : X \to X/\omega$ appelée projection canonique de ω associant chaque élément de X à sa classe d'équivalence par ω, c'est-à-dire $p_\omega(x) = [x]_\omega$ pour $x \in X$. Lorsque $p_\omega(x) = p_\omega(x')$ pour $x, x' \in X$, on dit que x est équivalent à x' *modulo* ω noté $x \equiv x'(mod\ \omega)$. Pour $\pi, \omega \in \mathcal{E}(X)$, on dit que π raffine ω, noté $\pi \leq \omega$, si

$$(\forall x, x' \mid x, x' \in X : x \equiv x'(mod\ \pi) \implies x \equiv x'(mod\ \omega)).$$

Le symbole $\leq$ définit donc un ordre sur $\mathcal{E}(X)$. On définit $\bot, \top \in \mathcal{E}(X)$, respectivement la borne inférieure et la borne supérieure de $\mathcal{E}(X)$, par $(\forall x, x' \mid x, x' \in X : x \equiv x'(mod\ \bot) \iff x = x')$ et $(\forall x, x' \mid x, x' \in X : x \equiv x'(mod\ \top))$. La paire $(\mathcal{E}(X), \leq)$ est un treillis complet (voir les notes de Wonham [62]).

Pour une fonction quelconque $f : X \to Y$, le noyau de la fonction dénoté $\ker f$ définit une relation d'équivalence sur X comme suit : pour $x, x' \in X$,

$$x \equiv x'(mod\ \ker f) \iff f(x) = f(x')$$

et on dit alors que x est équivalent à x' modulo $\ker f$.

2.3.1.2 Structures de contrôle

Dans le contexte théorique de Wong [55], la notion de familles de sous-langages contrôlables est indépendante (a priori) de la forme que peut prendre le contrôle, aussi y définit-on la notion de *structure de contrôle*. Pour un langage préfixe clos $L \subseteq \Sigma^*$ représentant le comportement libre d'un système ($L = L(\mathbf{G})$ pour un certain générateur $\mathbf{G}$ représentant un SED), on dit que la fonction $\mathcal{C} : \mathcal{F}_L \to \mathcal{P}^2(L)$ définit une *structure de contrôle sur* L si, pour $H \in \mathcal{F}_L$ arbitraire :

(1) $\mathcal{C}(H) \subseteq \mathcal{P}(H)$ est un demi-treillis supérieur par rapport à $\cup$ dans $\mathcal{P}(H)$;

(2) $\emptyset, H \in \mathcal{C}(H)$;

(3) $K \in \mathcal{C}(H) \Rightarrow \overline{K} \in \mathcal{C}(H)$;

(4) pour $F \in \mathcal{F}_L$ tel que $H \subseteq F$, alors $\mathcal{C}(F) \cap \mathcal{P}(H) \subseteq \mathcal{C}(H)$, avec égalité lorsque $H \in \overline{\mathcal{C}}(F)$, où $\overline{\mathcal{C}}(F) := \mathcal{F}_F \cap \mathcal{C}(F)$.

Alors $\mathcal{C}(H)$ associe à H (un sous-comportement de L) l'ensemble des sous-langages contrôlables qui lui correspond dans le système (L). On appelle aussi H un *sous-système* de L, mais il ne faut pas confondre avec la notion de *composante* illustrée à la section 2.1.

Soit $\mathcal{C}_1, \mathcal{C}_2 : \mathcal{F}_L \to \mathcal{P}^2(L)$ des structures de contrôle sur L. On dit que $\mathcal{C}_1$ raffine $\mathcal{C}_2$, noté $\mathcal{C}_1 \leq \mathcal{C}_2$, si

$$(\forall\, H \in \mathcal{F}_L)\ \mathcal{C}_2(H) \subseteq \mathcal{C}_1(H).$$

Ainsi $\mathcal{C}_1$ est plus fine que $\mathcal{C}_2$ si $\mathcal{C}_1$ contient plus de sous-langages contrôlables que $\mathcal{C}_2$ pour chaque sous-système.

Pour $F \in \mathcal{F}_L$, on définit un opérateur de synthèse (*control operator* dans [58])

$$\kappa_F : \mathcal{P}(F) \to \mathcal{C}(F) : H \mapsto \bigcup \{J \subseteq H \mid\ J \in \mathcal{C}(F)\}$$

pour tout $H \subseteq F$. Cet opérateur associe à H le plus grand sous-langage contrôlable de H dans F. Les axiomes (1) et (2) sur les structures de contrôle nous assurent que κ_F est bien défini ; $\kappa_F(\emptyset) = \emptyset$, $\kappa_F(F) = F$ et $\mathcal{C}(F)$ est l'ensemble des points fixes de κ_F. De plus κ_F est monotone quant à l'inclusion d'ensembles, idempotent et préserve la fermeture préfixée (voir Wong [55], page 34 et la proposition 12 en page 35). On définit aussi $\overline{\kappa}_F := \kappa_{F|\mathcal{F}_F}$ la restriction de κ_F à $\mathcal{F}_F$.

Pour lier la notion de familles de sous-langages contrôlables aux mécanismes du contrôle il est nécessaire de spécialiser la notion de structure de contrôle. On retrouve donc dans Wong [55] les notions de *structure de contrôle standard*, une sous-classe propre de l'ensemble des structures de contrôle, et de *structure de contrôle standard localement définissable*, une sous-classe propre de l'ensemble des structures de contrôle standards.

Pour L fermé (tel que défini précédemment), une fonction $\mathcal{C} : \mathcal{F}_L \rightarrow \mathcal{P}^2(L)$ possédant les propriétés suivantes

(1) $\emptyset, L \in \overline{\mathcal{C}}(L)$;

(2) $\overline{\mathcal{C}}(L)$ est un sous-treillis complet de $\mathcal{P}(L)$;

(3) $(\forall\, H \in \mathcal{F}_L)\, \mathcal{C}(H) = \{K \subseteq H \mid (\exists\, K' \mid K' \in \overline{\mathcal{C}}(L) : K = H \cap K')\}$;

est une structure de contrôle (d'après la proposition 13 dans [55]) et elle est dite standard. On dit par suite qu'une structure de contrôle standard est *localement définissable* si pour tout $K \in \mathcal{C}(L)$ et $s \in \overline{K}$,

$$L - pos_L(\{s\sigma \mid s\sigma \in L - \overline{K}\}) \in \mathcal{C}(L)$$

2.3.1.3 Technologies de contrôle

Les éléments du contrôle sont définis par la notion de *technologie de contrôle*. On dit qu'une fonction $\Sigma_c : L \rightarrow \mathcal{P}^2(\Sigma)$, où

(1) $(\forall\, s \in L)\, \emptyset \in \Sigma_c(s)$;

(2) $(\forall\, s \in L)\, \Sigma_c(s)$ est un sous-treillis complet de $\mathcal{P}(\Sigma)$;

pour $L \subseteq \Sigma^*$ fermé, définit une technologie de contrôle. Pour $s \in L$, $\Sigma_c(s)$ donne l'ensemble des configurations de contrôle disponibles pour la chaîne s dans L, sous forme d'ensembles d'événements que l'on peut inhiber simultanément à ce point. L'option de n'inhiber aucun événement à s (donc de permettre tout $\sigma \in \Sigma$ tel que $s\sigma \in L$) correspond à $\emptyset \in \Sigma_c(s)$ et doit toujours être disponible.

On remarque que lorsque l'alphabet Σ est partitionné en sous-ensembles d'événements contrôlables Σ_{con} et incontrôlables Σ_{unc}, alors $\Sigma_c(s) := \mathcal{P}(\Sigma_{con})$ pour tout $s \in L$ définit

une technologie de contrôle. Cette technologie de contrôle est désignée sous le nom de *technologie de contrôle standard*, et est la technologie de contrôle dans le contexte théorique usuel de SCT (contexte de Ramadge et Wonham [45]).

Pour $H \in \mathcal{F}_L$ on définit $\Sigma_c^H := \Sigma_c|_H$ et

$$\Sigma_u^H : H \to \mathcal{P}^2(\Sigma) : s \longmapsto \{\Sigma - X \mid X \in \Sigma_c^H(s)\}$$

pour $s \in H$. Alors $\Sigma_u^L(s)$ donne les configurations de contrôle disponibles à s dans L sous forme d'ensembles d'événements permis à ce point. Lorsque $H = L$ on omet généralement la désignation en exposant et $\Sigma_u^L(s) = \Sigma_u(s)$.

Lorsque L est ainsi muni d'une technologie de contrôle, pour $H \in \mathcal{F}_L$ et $K \subseteq H$, on dit que K est contrôlable par rapport à H si pour tout $s \in \overline{K}$

$$(\exists\, X \in \Sigma_u^H(s))\ X \cap Elig_H(s) = Elig_K(s).$$

Ainsi K est contrôlable par rapport à H si l'ensemble des événements pouvant être générés à la suite de s peut être restreint par un contrôle approprié ($X \in \Sigma_u^H(s)$) à l'ensemble des événements prolongeant s dans $\overline{K}$. Sur cette base, en définissant $\mathcal{C} : \mathcal{F}_L \to \mathcal{P}^2(L)$ par

$$\mathcal{C}(H) := \{K \subseteq H \mid K \text{ est contrôlable par rapport à } H\}$$

et

$$\overline{\mathcal{C}}(H) := \mathcal{C}(H) \cap \mathcal{F}_H$$

les propositions 15 à 19 de Wong [55] démontrent que $\mathcal{C}$ est une structure de contrôle standard localement définissable. Ainsi, pour $L \subseteq \Sigma^*$, il suffit que L soit muni d'une technologie de contrôle pour en déduire qu'il existe $\mathcal{C} : \mathcal{F}_L \to \mathcal{P}^2(L)$ une structure de contrôle standard localement définissable sur L. On dit qu'une technologie de contrôle pour L *induit* une structure de contrôle standard localement définissable sur L. De façon duale, les propositions 20 et 21 de Wong [55] établissent que s'il existe $\mathcal{C} : \mathcal{F}_L \to \mathcal{P}^2(L)$ une structure de contrôle standard localement définissable sur $L \subseteq \Sigma^*$ fermé, alors

$$\Sigma_c : L \to \mathcal{P}^2(\Sigma) : s \longmapsto \{\{\sigma \in \Sigma \mid s\sigma \in L - \overline{K}\} \mid K \in \mathcal{C}(L),\ s \in \overline{K}\}$$

définit une technologie de contrôle pour L.

Il existe donc un lien fort entre *technologie de contrôle* d'une part et *structure de contrôle standard localement définissable* d'autre part. On se rappelle qu'un système (ici un SED) est modélisé par une paire (L, L_m), où $L = \overline{L_m}$. Dans un système, une technologie de contrôle *induit* toujours une structure de contrôle standard localement définissable (sur L), et réciproquement, pour un système muni d'une structure de contrôle standard localement définissable il existe toujours une technologie de contrôle qui l'implémente.

2.3.1.4 Existence de lois de contrôle

Pour $C : \mathcal{F}_L \to \mathcal{P}^2(L)$ une structure de contrôle standard localement définissable dotée d'une technologie de contrôle $\Sigma_c : L \to \mathcal{P}^2(\Sigma)$ correspondante, on dit qu'une fonction $V : L \to \mathcal{P}(\Sigma)$, telle que $V(s) \in \Sigma_u(s)$ pour $s \in L$ arbitraire, définit une loi de contrôle sur L.

Le comportement de L sous le contrôle de V est un langage $L(V) \subseteq L$ défini par

(i) $\varepsilon \in L(V)$;

(ii) $s\sigma \in L(V)$ ssi $s \in L(V)$, $\sigma \in V(s)$ et $s\sigma \in L$.

Pour (L, L_m) le modèle linguistique d'un système, le langage marqué de $L(V)$ est défini par $L_m(V) = L(V) \cap L_m$, et V est non bloquante si $\overline{L_m(V)} = L(V)$.

Proposition 85. *[55] Soit $K \subseteq L_m$, $K \neq \emptyset$. Alors il existe une loi de contrôle non bloquante V sur L telle que $L_m(V) = K$ si et seulement si*

(i) K est contrôlable par rapport à L ;

(ii) K est L_m-fermé.

2.3.1.5 Liens avec la théorie classique

Il est intéressant de remarquer que selon Wong et Wonham [59], dans le contexte usuel de SCT, c'est-à-dire $L \subseteq \Sigma^*$ fermé où $\Sigma = \Sigma_{con} \dot{\cup} \Sigma_{unc}$, alors pour $H \in \mathcal{F}_L$

$$C(H) := \{K \subseteq H \mid \overline{K}\Sigma_{unc} \cap H \subseteq \overline{K}\}$$

définit une structure de contrôle standard localement définissable ayant $\Sigma_c(s) := \mathcal{P}(\Sigma_{con})$ pour tout $s \in L$ (la technologie de contrôle standard) comme technologie de contrôle

associée. Aussi, dans ce même contexte, d'après Wong ([55], section 3.1), l'opérateur de synthèse est alors défini par :

$$\kappa_H(E) = \sup\mathcal{C}_H(E). \text{ le plus grand sous-langage de } E \text{ contrôlable par rapport à } H$$

pour $E \subseteq H \subseteq L$. Ainsi dans le contexte théorique de Wong, les résultats de SCT concernant la synthèse sont tous applicables moyennant que le système concerné soit muni d'une technologie de contrôle standard.

2.3.2 Modèle du contrôle hiérarchique

La figure 2.1 (due à [62]) présente de façon schématique les éléments du contrôle hiérarchique. Dans cette figure $\mathbf{G}_{lo}$ représente le système à contrôler par l'intermédiaire de $\mathbf{C}_{lo}$, l'opérateur. $\mathbf{G}_{hi}$ est un modèle abstrait ou simplifié de $\mathbf{G}_{lo}$ utilisé par $\mathbf{C}_{hi}$, le gestionnaire, pour prendre des décisions de contrôle. $\mathbf{G}_{hi}$ suit l'évolution de $\mathbf{G}_{lo}$ par l'intermédiaire d'un canal d'information $\mathbf{inf}_{lohi}$, lui permettant de mettre à jour son propre état pour refléter fidèlement l'évolution de $\mathbf{G}_{lo}$ à tout moment. Dans le système de bas niveau, la relation de contrôle entre $\mathbf{G}_{lo}$ et $\mathbf{C}_{lo}$ est conforme au modèle de SCT, où $\mathbf{inf}_{lo}$ procure l'information sur l'occurrence d'événements dans $\mathbf{G}_{lo}$ et $\mathbf{con}_{lo}$ la rétroaction. L'abstraction $\mathbf{G}_{hi}$ est aussi dotée d'une structure de contrôle conforme au modèle de SCT, de sorte que $\mathbf{C}_{hi}$ pourrait, sur la base d'information recueillie sur $\mathbf{inf}_{hi}$, produire une rétroaction sur $\mathbf{con}_{hi}$ pour contrôler $\mathbf{G}_{hi}$. Mais dans les faits, le comportement de $\mathbf{G}_{hi}$ est complètement déterminé par l'information lue sur $\mathbf{inf}_{lohi}$ et générée par $\mathbf{G}_{lo}$, le lien $\mathbf{con}_{hi}$ n'est que virtuel. La rétroaction calculée par $\mathbf{C}_{hi}$ est plutôt transmise à $\mathbf{C}_{lo}$, sur le lien $\mathbf{com}_{hilo}$, qui doit la traduire de manière à ce que $\mathbf{G}_{lo}$ assure adéquatement l'objectif de contrôle, fermant ainsi la boucle de rétroaction du système global.

2.3.2.1 Projections causales

Pour définir le contexte formel de la figure 2.1, posons Σ et T les alphabets respectifs de l'agent et de l'abstraction, où $\Sigma \cap T = \emptyset$. Posons (L, L_m) comme modèle linguistique pour l'agent avec $L = L(\mathbf{G}_{lo})$ fermé, représentant le comportement généré de l'agent. Le canal

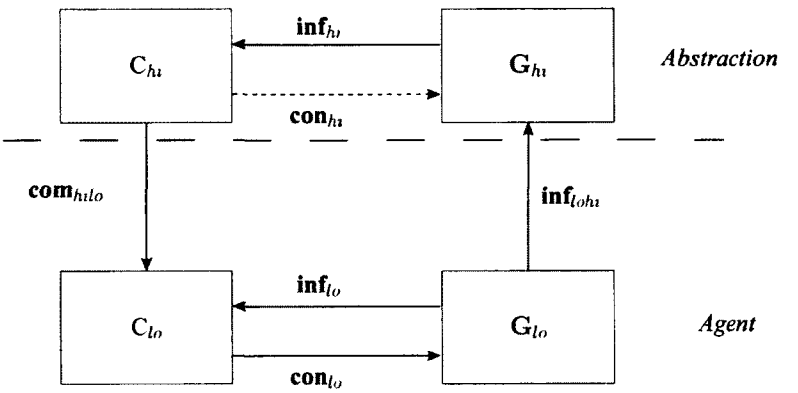


FIGURE 2.1 – Structure d'un contrôle hiérarchique

d'information $\mathbf{inf}_{lohi}$ est alors modélisé par une fonction $\theta : L \rightarrow T^*$ ayant la propriété de conserver les préfixes, c'est-à-dire

$$(\forall K \subseteq L)\ \theta(\overline{K}) = \overline{\theta(K)}.$$

Ce type de fonction est équivalent ([58], proposition 3) à une *projection causale* (définie dans [66, 62]) où, avec $\omega : L \rightarrow T$ une fonction partielle arbitraire,

$$\begin{aligned} \theta(\varepsilon) &= \varepsilon \\ \theta(s\sigma) &= \begin{cases} \theta(s) \cdot \omega(s\sigma), & \text{si } \omega(s\sigma) \text{ est définie} \\ \theta(s), & \text{autrement} \end{cases} \end{aligned}$$

pour $s \in L$ et $\sigma \in \Sigma$. Ce type de projection préserve l'historique du système de bas niveau dans le sens que pour $s, u \in L$, si $s \leq u$ alors $\theta(s) \leq \theta(u)$.

La projection $\theta : L \rightarrow T^*$ permet de définir le modèle linguistique de l'abstraction (M, M_m), où $M := \theta(L) \subseteq T^*$ fermé représente le comportement généré de l'abstraction $(M = L(\mathbf{G}_{hi}))$ et $M_m := \theta(L_m)$ son comportement marqué. Par suite il est plus commode

de redéfinir la fonction θ comme $\theta : L \to M$ en restreignant le co-domaine de la fonction θ originale à M. Avec θ ainsi redéfinie, le canal $\mathbf{com}_{hlo}$ est modélisé par la fonction de préimage par θ, c'est-à-dire

$$\theta^{-1} : \mathcal{P}(M) \to \mathcal{P}(L) : N \longmapsto \theta^{-1}(N) := \{s \in L \mid \theta(s) \in N\}.$$

Cette dernière fonction a, entre autres, la propriété que

$$(\forall N \in \mathcal{F}_M)\ \theta^{-1}(N) \in \mathcal{F}_L$$

([58], proposition 4), la préimage par θ d'un sous-langage fermé de M est donc un sous-langage fermé de L. On peut trouver quelques propriétés utiles des projections causales en annexe B.

Il est à noter qu'il existe à proprement parler plusieurs définitions de la fonction de projection. À l'instar de Wong et Wonham [59] on peut définir

- $\theta : L \to M$, associant une chaîne de L à une chaîne de M (la définition vue plus haut) ;
- $\theta_* : \mathcal{P}(L) \to \mathcal{P}(M) : H \longmapsto \theta_*(H) := \{\theta(s) \mid s \in H\}$, associant un sous-langage de L à un sous-langage de M ;
- $\theta_{**} : \mathcal{P}^2(L) \to \mathcal{P}^2(M) : X \longmapsto \theta_{**}(X) := \{\theta_*(H) \mid H \in X\}$, associant une famille de sous-langages de L à une famille de sous-langages de M.

Il devrait être évident que toutes ces fonctions sont surjectives. En général on n'utilise pas les notations θ_* et θ_{**} à moins que le contexte porte à confusion, par exemple pour une structure de contrôle $\mathcal{C}_{lo}$ sur L, $\theta(\mathcal{C}_{lo}(L))$ signifie généralement $\theta_{**}(\mathcal{C}_{lo}(L))$. Cet usage peut cependant porter à confusion concernant la définition à utiliser pour θ^{-1} dans certains contextes.

Notons donc que la fonction de préimage par θ, c'est-à-dire $\theta^{-1} : \mathcal{P}(M) \to \mathcal{P}(L)$, est utilisée sur des sous-langages de M et aussi, par abus de notation, pour des chaînes $t \in M$, par exemple $\theta^{-1}(t) = \theta^{-1}(\{t\})$. Lorsque θ^{-1} est utilisée sur des familles de sous-langages de M on utilise

$$(\theta^{-1})_* : \mathcal{P}^2(M) \to \mathcal{P}^2(L) : X \longmapsto (\theta^{-1})_*(X) := \{\theta^{-1}(N) \mid N \in X\}$$

par exemple $\theta^{-1}(\mathcal{C}_{hi}(M)) = (\theta^{-1})_*(\mathcal{C}_{hi}(M)) := \{\theta^{-1}(N) \mid N \in \mathcal{C}_{hi}(M)\}$.

On définit aussi, associée à $\theta : L \to M$, une fonction partielle $\omega : L \to T$ appelée *fonction de sortie*, où

$$\omega(s\sigma) := \begin{cases} \tau, & \text{si } \theta(s\sigma) = \theta(s)\tau\,; \\ \textit{indéfinie}. & \text{autrement.} \end{cases}$$

pour $s\sigma \in L$. La fonction ω n'est donc définie que pour les chaînes $s\sigma \in L$ provoquant la génération d'un événement à travers θ. On appelle de telles chaînes, des chaînes *vocales* (voir [62]). On aura donc aussi $\omega^{-1} : \mathcal{P}(T) \to \mathcal{P}(L)$ et

$$\omega^{-1}(X) := \{s \in L \mid \omega(s)! \wedge \omega(s) \in X\}$$

pour $X \subseteq T$ ($\omega(s)!$ dénote que $\omega(s)$ est définie). Aussi $\omega^{-1}(\tau) = \omega^{-1}(\{\tau\})$, par abus de notation, dénote l'ensemble des chaînes de L générant l'événement τ à travers θ.

Aussi on définit

$$L_{voc} := \omega^{-1}(T) \cup \{\varepsilon\}$$

l'ensemble des chaînes *vocales* de L, c'est-à-dire l'ensemble des chaînes de L qui provoquent la génération d'un événement à travers θ auquel on inclut la chaîne vide.

Finalement, pour $\theta : L \to M$, on définit aussi $\theta_v := \theta_{|L_{voc}}$, la restriction de θ à L_{voc}, et θ_v^{-1} par

$$\theta_v^{-1}(N) := \theta^{-1}(N) \cap L_{voc}$$

pour $N \subseteq M$ arbitraire.

2.3.2.2 Consistance du contrôle

En considérant L_m, L, θ, θ^{-1}, M_m et M tels que définis précédemment, on pose $\mathcal{C}_{lo}$ une structure de contrôle sur L. On dit que le triplet $(L, \theta, \mathcal{C}_{lo})$ est doté de la propriété de *consistance du contrôle* s'il existe une fonction $\mathcal{C}_{hi} : \mathcal{F}_M \to \mathcal{P}^2(M)$ telle que

$$(\forall\, N \in \mathcal{F}_M)\ \mathcal{C}_{hi}(N) = \theta(\mathcal{C}_{lo}(H)),$$

où $H := \theta^{-1}(N)$. La fonction C_{hi} n'existe que si et seulement si $\ker \theta \leq \ker(\theta \circ C_{lo})$ et puisque $\theta : \mathcal{F}_L \to \mathcal{F}_M$ est surjective, C_{hi} est unique si elle existe. Dans ces conditions $C_{hi} : \mathcal{F}_M \to \mathcal{P}^2(M)$ est une structure de contrôle sur M ([58], proposition 5).

Le théorème 1 de Wong et Wonham [58] met en relation la propriété de consistance du contrôle et celle de *consistance hiérarchique* exposée dans Zhong et Wonham [66]

$$C_{hi}(N) = \theta(C_{lo}(H)) \iff \kappa_N = \theta \circ \kappa_H \circ \theta^{-1}$$

pour $N \in \mathcal{F}_M$ et $H := \theta^{-1}(N)$, en lui donnant une définition algébrique précise. On se rappelle que κ_N et κ_H sont les opérateurs de synthèse liés respectivement à $C_{hi}(N)$ et $C_{lo}(H)$. Dans la figure 2.1, on peut dire que κ_M modélise $\mathbf{con}_{hi}$ (le contrôle exercé par l'élément C_{hi}) et κ_L modélise $\mathbf{con}_{lo}$ (le contrôle exercé par l'élément C_{lo}). En se rappelant que θ^{-1} modélise $\mathbf{com}_{hilo}$ et θ modélise $\mathbf{inf}_{lohi}$, on a informellement

$$\mathbf{con}_{hi} \equiv \mathbf{inf}_{lohi} \circ \mathbf{con}_{lo} \circ \mathbf{com}_{hilo}.$$

Ainsi, dans la figure 2.1, pour un objectif de contrôle $E \subseteq M$, l'effet de l'élément C_{hi} est de synthétiser $\kappa_M(E)$ sur G_{hi} (l'abstraction). Cet effet peut être obtenu à travers l'agent en lui soumettant $\theta^{-1}(E)$ comme objectif de contrôle à travers $\mathbf{com}_{hilo}$. L'effet de l'élément C_{lo} est alors de synthétiser $\kappa_L \circ \theta^{-1}(E)$ sur G_{lo}, dont la projection $\theta(\kappa_L \circ \theta^{-1}(E))$ est parfaitement équivalente à $\kappa_M(E)$, fermant ainsi la boucle. Ainsi on a formellement $\kappa_M = \theta \circ \kappa_L \circ \theta^{-1}$, c'est-à-dire la consistance hiérarchique pour le triplet (L, M, θ).

Tel que mentionné dans [55] (section 6.2), le problème de déterminer si un triplet (L, θ, C_{lo}) présente la consistance du contrôle (c'est-à-dire si $\ker \theta \leq \ker(\theta \circ C_{lo})$) est généralement indécidable. Or en contrôle hiérarchique, la structure de contrôle de l'abstraction C_{hi} n'existe que si et seulement si $\ker \theta \leq \ker(\theta \circ C_{lo})$. Il est donc généralement nécessaire de se rabattre sur des conditions suffisantes pour assurer la consistance du contrôle. Ces conditions sont généralement exprimées sous forme de propriétés de la projection θ assurant, entre autres choses, la consistance du contrôle. Le lecteur est invité à consulter Wong [55] (section 6.4) et Wong et Wonham [59] (appendices A et B) pour des exemples d'investigations substantielles dans cette direction.

Il est intéressant de noter qu'il existe une forme affaiblie de la consistance du contrôle correspondant au théorème 2 de Wong et Wonham [58], notamment

$$\mathcal{C}_{hi}(N) \subseteq \theta(\mathcal{C}_{lo}(H)) \iff \theta \circ \kappa_H \circ \theta^{-1}|_{\mathcal{C}_{hi}(N)} = id_{\mathcal{C}_{hi}(N)}$$

pour $N \in \mathcal{F}_M$ et $H := \theta^{-1}(N)$. La consistance du contrôle n'est alors assurée que si l'objectif de contrôle soumis à l'agent est $\theta^{-1}(\kappa_M(E))$, pour $E \subseteq M$.

2.3.3 La propriété d'observateur d'une projection

La définition de la propriété d'observateur fait appel à un appareillage théorique qu'il serait trop long de reproduire ici. Aussi ce qui suit n'est-il qu'un survol explicatif. Le lecteur est invité à consulter les travaux de Wong pour une argumentation complète sur le sujet. La notion d'*observateur* en tant qu'équivalence observationnelle entre systèmes dynamiques origine de Wonham [61]. La thèse de doctorat de Wong [55], dont les articles [58, 59] présentent un sommaire condensé, fournit une argumentation détaillée du sujet. Par suite Wong et al [57] ainsi que Wong et Wonham [60, 62] font le lien avec la notion de quasi-congruence et avec le contexte des générateurs finis.

La consistance du contrôle permet de garantir, dans le contexte du contrôle hiérarchique (figure 2.1), qu'une synthèse dans l'abstraction correspond exactement (du point de vue du contenu des langages synthétisés) à une synthèse équivalente dans l'agent. De ce point de vue, il est garanti que tout langage contrôlable de l'abstraction peut être implémenté à travers l'agent. Rien ne garantit cependant qu'une synthèse non bloquante dans l'abstraction correspond nécessairement à une synthèse non bloquante dans l'agent. Pour obtenir cette assurance il faut concevoir la projection de façon à ce qu'elle soit dotée d'une propriété appelée *propriété d'observateur.*

La figure 2.2 illustre informellement le problème. Dans cette figure le modèle pour L indique en fait la dynamique de la paire (L, θ). Les transitions entre états sont étiquetées par des expressions de la forme α/τ signifiant que la génération de α par l'agent provoque la génération d'un événement abstrait τ, ou encore que $\theta(\alpha) = \tau$. Le modèle pour M n'est qu'un simple générateur pour le langage. Les états sont étiquetés par les chaînes respectivement générées (l'historique) dans chaque système à chaque point de leur évolution. La

technologie de contrôle de L (l'agent) est standard et tous ses événements sont contrôlables ce qui induit une technologie de contrôle standard dans M (l'abstraction) où tous les événements abstraits sont aussi contrôlables. Puisque tout est contrôlable, il est aisé de vérifier qu'il y a consistance du contrôle. On a donc $L = \{\varepsilon, \alpha, \beta, \alpha\gamma, \beta\delta\}$, $L_m = \{\alpha\gamma, \beta\delta\}$ et $M = \{\varepsilon, \tau, \tau\tau_1, \tau\tau_2\}$, $M_m = \{\tau\tau_1, \tau\tau_2\}$.

Soit donc $E = \{\tau\tau_1\} \subseteq M$ un objectif de contrôle. Alors $\kappa_M(E) = E$ est M_m-fermé c'est-à-dire que $E = \overline{E} \cap M_m = \{\varepsilon, \tau, \tau\tau_1\} \cap \{\tau\tau_1, \tau\tau_2\}$ et donc non bloquant, c'est-à-dire que $\overline{E} = \overline{\overline{E} \cap M_m}$. Par consistance du contrôle on a que $\kappa_M(E) = \theta \circ \kappa_L \circ \theta^{-1}(E)$. Dans ce contexte, tout se passe comme si on synthétisait directement la préimage de l'objectif de contrôle $\theta^{-1}(E)$ sur l'agent, et on peut vérifier que $K' := \kappa_L \circ \theta^{-1}(E) = \{\alpha\gamma\}$ est non bloquant dans l'agent c'est-à-dire $\overline{K' \cap L_m} = \overline{\{\varepsilon, \alpha, \alpha\gamma\} \cap \{\alpha\gamma, \beta\delta\}} = \{\varepsilon, \alpha, \alpha\gamma\} = \overline{K'}$.

Cependant la stratégie du contrôle hiérarchique est indirecte ; elle vise à implémenter une loi de contrôle V_Σ dans l'agent sur la base de la loi de contrôle obtenue pour l'abstraction V_T en utilisant une règle telle que celle utilisée dans Wong et Wonham [59] (proposition 12 en appendice C). Le résultat de cette manoeuvre équivaut (du point de vue du contenu des langages concernés) à utiliser $\theta^{-1}(\overline{\kappa_M(E)})$ comme spécification pour une synthèse sur l'agent (voir à cet effet [59], postulat à la proposition 12). On vise donc à implémenter le langage $K := \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)})$ sur l'agent, sachant que sa projection $\theta \circ \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) = \kappa_M(\overline{\kappa_M(E)}) = \overline{\kappa_M(E)}$ est précisément $L(V_T/\mathbf{G}_h)$, le comportement sous contrôle du système de haut niveau. Mais voilà, dans l'exemple de la figure 2.2, $K := \{\varepsilon, \alpha, \beta, \alpha\gamma\}$ et ce langage est bloquant à l'état β dans l'agent. Ici β se retrouve dans la spécification parce que $\theta(\alpha)\tau_1 = \tau\tau_1 \in \kappa_M(E)$ et donc $\tau \in \overline{\kappa_M(E)}$. Ainsi

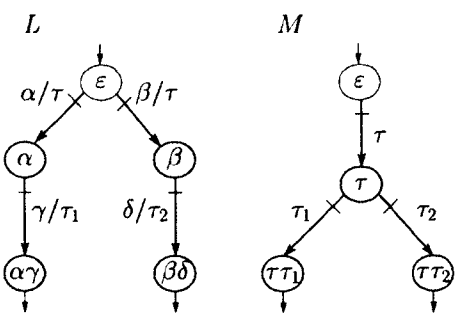


FIGURE 2.2 – Contrôle hiérarchique bloquant

$\theta^{-1}(\overline{\kappa_M(E)}) \supseteq \theta^{-1}(\tau) = \{\alpha, \beta\}$ et les deux chaînes partagent la même classe d'équivalence de $\ker \theta$. Or il n'y a pas de chaîne $s \in L$ où $\beta \leq s$ et $\theta(s) = \tau\tau_1$ et l'agent bloque. Une façon de remédier à ce problème est de construire θ (ou de la raffiner) de sorte que toutes les chaînes de L partageant une même classe d'équivalence de $\ker \theta$ aient le même ensemble de développements futurs à travers θ. Ceci est l'essence de la propriété d'observateur.

2.3.3.1 Quasi-congruences et projections causales

Il existe une caractérisation alternative de *projection causale* en plus de celle donnée en section 2.3.2.1. Cette caractérisation fait appel à la notion de *quasi-congruence*.

Pour deux ensembles X et Y, une fonction $f : X \to \mathcal{P}(Y)$ et une relation d'équivalence $\rho \in \mathcal{E}(Y)$, on définit $\rho \circ f \in \mathcal{E}(X)$ par

$$x \equiv x'(mod\ \rho \circ f) \iff p_\rho(f(x)) = p_\rho(f(x'))$$

pour $x, x' \in X$, et où $p_\rho : \mathcal{P}(Y) \to \mathcal{P}(Y/\rho) : Y' \longmapsto \{[y]_\rho \mid y \in Y'\}$ est l'extension standard de la projection canonique par ρ dont le domaine a été étendu à des ensembles. Ainsi $\rho \circ f = \ker(p_\rho \circ f)$ car pour $g : X \to Y$, $x \equiv x'(mod\ \rho \circ g)$ si et seulement si $g(x) \equiv g(x')(mod\ \rho)$.

Pour X un ensemble et $\alpha_i : X \to \mathcal{P}(X)$ un ensemble de fonctions d'index $i \in I$, on appelle la structure $\mathcal{S} := (X, \{\alpha_i \mid i \in I\})$ un système dynamique (*non deterministic dynamic system* dans [62]). Dans un tel système, les éléments de l'ensemble X sont considérés représenter les *états* du système et l'ensemble des $\{\alpha_i \mid i \in I\}$ représente la *dynamique* du système (l'effet de sa fonction de transition). Par convention on étend généralement le domaine de α_i à des ensembles $\alpha_i(X') := \bigcup_{x \in X'} \alpha_i(x)$ pour $X' \subseteq X$. On dit que $\omega \in \mathcal{E}(X)$ est une *quasi-congruence* pour $\mathcal{S}$ si

$$\omega \leq \bigwedge_{i \in I} (\omega \circ \alpha_i).$$

Pour $L \subseteq \Sigma^*$ un langage arbitraire (pas nécessairement fermé) sur un alphabet Σ, en définissant $pre : \Sigma^* \to \mathcal{P}(\Sigma^*) : s \longmapsto \overline{\{s\}}$ l'opérateur de fermeture préfixée, on a le résultat suivant.

Proposition 2. *[57] Si* $\theta : \overline{L} \to T^*$ *est une projection causale, alors* $\ker \theta \leq (\ker \theta) \circ pre$. *Symétriquement, pour* $\pi \in \mathcal{E}(\overline{L})$ *telle que* $\pi \leq \pi \circ pre$, *il existe un alphabet d'événements* T *et une projection causale* $\theta : \overline{L} \to T^*$ *pour laquelle* $\ker \theta = \pi$.

Autrement dit, dans des termes légèrement différents ([60], proposition 2.1), la projection $\theta : \overline{L} \to T^*$ n'est causale que si $\ker \theta$ est une quasi-congruence pour le système dynamique $(\overline{L}, pre)$. Et réciproquement s'il existe une quasi-congruence $\pi \in \mathcal{E}(\overline{L})$ pour le système dynamique $(\overline{L}, pre)$ alors il existe T et une projection causale $\theta : \overline{L} \to T^*$ tels que $\ker \theta = \pi$.

2.3.3.2 Modèle d'une projection causale

Il est souvent utile de modéliser une projection causale sous forme d'une machine de Mealy (voir [31]). Rappelons qu'une machine de Mealy déterministe est une structure algébrique (un 7-uplet)

$$\mathbf{A} = (X, \Sigma, T, \xi, \phi, x_0, X_m)$$

où X est l'ensemble d'états, Σ l'alphabet, T un *alphabet de sortie*, $\xi : X \times \Sigma \to X$ la fonction (partielle) de transition, $\phi : X \times \Sigma \to T$ une fonction (partielle) de *sortie*, x_0 son état initial et X_m l'ensemble de ses états marqués. On note tout de suite que l'AFD

$$\mathbf{A_{ts}} = (X, \Sigma, \xi, x_0, X_m)$$

constitue la structure de transition (*transition structure* dans [60]) de la machine de Mealy. Naturellement il est possible de spécifier des machines de Mealy dont la structure de transition n'est ni déterministe ni finie.

Pour modéliser une projection causale $\theta : L \to M$ ($L \subseteq \Sigma^*$ et $M \subseteq T^*$ fermés avec $\Sigma \cap T = \emptyset$) par une machine de Mealy, il est commode de procéder à partir d'un générateur

$$\mathbf{G_{ts}} = (Q, \Sigma, \delta, q_0, Q_m) \tag{2.1}$$

où $L = L(\mathbf{G_{ts}})$.

On adjoint à $\mathbf{G_{ts}}$ un alphabet $T \cup \{\tau_0\}$ et une fonction de sortie $\lambda : Q \times \Sigma \to T \cup \{\tau_0\}$ définie par

$$\lambda(s,\sigma) := \begin{cases} \tau_0, & \text{si } \theta(s\sigma) = \theta(s) \text{ et } s\sigma \in L\,; \\ \tau, & \text{si } \theta(s\sigma) = \theta(s)\tau \text{ et } s\sigma \in L\,; \\ \textit{indéfinie}, & \text{autrement.} \end{cases} \tag{2.2}$$

Ici l'occurrence de l'événement $\tau_0 \notin T \cup \Sigma$ à la sortie de la machine de Mealy signifie que θ efface l'événement correspondant à une certaine transition dans $\mathbf{G_{ts}}$, et cette transition est alors dite *inobservable* à travers θ (*non vocal transition* par opposition à *vocal transition* dans [58]). On forme ainsi

$$\mathbf{G} = (Q, \Sigma, T \cup \{\tau_0\}, \delta, \lambda, q_0, Q_m) \tag{2.3}$$

la machine de Mealy *modélisant* la paire (L, θ) et ayant $\mathbf{G_{ts}}$ comme structure de transition.

2.3.3.3 Système dynamique correspondant à une projection causale

Pour exprimer le système dynamique correspondant à (L, θ) il est nécessaire d'incorporer le comportement de la fonction de sortie de la machine de Mealy dans sa structure de transition. Pour ce faire on a besoin de la notion de *réétiquetage d'un automate.*

Soit $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ un automate potentiellement non déterministe; donc on a $\delta : Q \times \Sigma \to 2^Q$. L'ensemble des transitions de $\mathbf{G}$ est alors donné par

$$\mathcal{T}_\delta := \{(q, \sigma, q') \in Q \times \Sigma \times Q \mid\ q' \in \delta(q, \sigma)\}.$$

On dit qu'un automate $\mathbf{G}' = (Q, \Sigma', \delta', q_0, Q_m)$, (potentiellement non déterministe) constitue un réétiquetage de $\mathbf{G}$ s'il existe une fonction surjective (dite fonction de réétiquetage)

$$r : \mathcal{T}_\delta \to \mathcal{T}_{\delta'} : (q, \sigma, q') \mapsto (q, \sigma', q')$$

où $\sigma' \in \Sigma'$, transformant $\mathbf{G}$ en $\mathbf{G}'$. Si $\mathbf{G}'$ est déterministe, alors $\mathbf{G}'$ constitue un réétiquetage déterministe de $\mathbf{G}$.

Pour une machine de Mealy G (définie comme dans l'équation 2.3) modélisant la paire (L, θ), on définit

$$\widehat{\mathbf{G}} = (Q, \Sigma \cup T, \widehat{\delta}, q_0, Q_m) \tag{2.4}$$

un réétiquetage de sa structure de transition ($\mathbf{G_{ts}}$) avec $r_{\mathbf{G}} : \mathcal{T}_\delta \to \mathcal{T}_{\widehat{\delta}}$ définie comme suit :

$$r_{\mathbf{G}}((q, \sigma, q')) := \begin{cases} (q, \tau, q'), & \text{si } \lambda(q, \sigma) = \tau \in T\,; \\ (q, \sigma, q'), & \text{si } \lambda(q, \sigma) = \tau_0. \end{cases}$$

Ainsi $r_{\mathbf{G}}$ remplace, dans $\mathbf{G_{ts}}$, l'étiquette de toutes les transitions observables à travers θ par le symbole généré par la fonction de sortie de la machine de Mealy lors de cette transition. Il se peut que $\widehat{\mathbf{G}}$ soit non déterministe.

On peut alors exprimer le système dynamique (non déterministe) correspondant à $\widehat{\mathbf{G}}$ en définissant

$$\Delta_\tau : Q \to 2^Q : q \longmapsto \bigcup \{\, \widehat{\delta}(q, u\tau u') \mid uu' \in (\Sigma - T)^* \,\}$$

pour chaque $\tau \in T$ et

$$\Delta_m : Q \to 2^{Q_m} : q \longmapsto \bigcup \{\, \widehat{\delta}(q, u) \cap Q_m \mid u \in (\Sigma - T)^* \,\}$$

et on obtient l'expression du système dynamique

$$\widetilde{\mathbf{G}} := (Q, \{\Delta_\tau \mid \tau \in T\} \cup \{\Delta_m\}, q_0, Q_m) \tag{2.5}$$

en supposant que $m \notin T$. Ici $\Delta_m(q) \subseteq Q_m$ représente l'ensemble des états finaux que l'on peut atteindre à partir de $q \in Q$ dans $\widehat{\mathbf{G}}$, en empruntant seulement des transitions inobservables à travers θ. De même, $\Delta_\tau(q) \subseteq Q$ représente l'ensemble des états que l'on peut atteindre à partir de $q \in Q$ dans $\widehat{\mathbf{G}}$, en empruntant uniquement que des chemins ne contenant que des transitions inobservables à travers θ en plus d'une et une seule transition observable à travers θ d'étiquette τ.

Cette définition du *système dynamique* correspondant à une structure de transition est conforme à la notion de *système dynamique* donnée précédemment ; la présence d'un état initial q_0 et d'un ensemble d'états marqués Q_m ne change rien de fondamental. On peut

étendre la définition de Δ_τ aux chaînes $t\tau \in T^*$ de la façon suivante :

$$\begin{aligned} \Delta_\varepsilon(q) &= \{q\} \\ \Delta_{t\tau}(q) &= \bigcup\{\Delta_\tau(q') \mid q' \in \Delta_t(q)\} \end{aligned}$$

pour $q \in Q$.

2.3.3.4 Caractérisation exhaustive d'une projection causale

Pour lier l'argumentation présentée par Wong et Wonham [58] (et aussi [55] et [62]) à la présentation du concept d'observateur par Wong et Wonham [60], il peut être utile de considérer certaines des différences apparentes entre les deux argumentaires. Dans [60] en section 2.4.2, vers la fin, on y fait la remarque suivante :

> One of the objectives of the paper is to define a computation on a (finite) Mealy automaton G producing a causal reporter map θ^* which has π^* as its equivalence kernel. To do this, we need to relate the computation on G to π^*, which is an equivalence relation on $\overline{L}$.

Ce qui suit (dans [60]) présente un intérêt certain aux lecteurs de [58].

On définit la structure de transition $\mathbf{T_{ts}} = (X, \Sigma, \xi, x_0, X_m)$, un automate déterministe (dont l'espace des états n'est pas nécessairement fini) pour lequel il existe une bijection $f : L \to X$, où $L = \overline{L_m} \subseteq \Sigma^*$, et telle que

1. $f(\varepsilon) = x_0$;
2. $f(L_m) = X_m$;
3. pour $s \in L$ et $\sigma \in \Sigma$, $\xi(f(s), \sigma)$ est définie si et seulement si $s\sigma \in L$ et dans ce cas $\xi(f(s), \sigma) = f(s\sigma)$.

Ensuite on adjoint à $\mathbf{T_{ts}}$ l'alphabet de sortie $T \cup \{\tau_0\}$ et la fonction de sortie $\lambda^\circ : X \times \Sigma \to T \cup \{\tau_0\}$ induite par $\lambda^\circ(x, \sigma) = \lambda(f^{-1}(x), \sigma)$, où λ est telle que définie dans l'équation 2.2 et f^{-1} est l'inverse de la bijection f. Par abus de notation on utilisera λ pour λ° quand le contexte sera clair, et la fonction de transition est étendue aux chaînes $\xi : X \times \Sigma^* \to X$ de

façon standard (voir section 1.1.3) au besoin. On obtient donc la machine de Mealy

$$\mathbf{T} = (X, \Sigma, T \cup \{\tau_0\}, \xi, \lambda, x_0, X_m)$$

et il devrait être clair que le graphe de $\mathbf{T}$ est un arbre (possiblement infini) tel qu'illustré en figure 2.3. Comme dans l'équation 2.4, pour $\mathbf{T_{ts}}$, on obtient

$$\widehat{\mathbf{T}} = (X, \Sigma \cup T, \widehat{\xi}, x_0, X_m)$$

et le système dynamique

$$\widetilde{\mathbf{T}} = (X, \{\Xi_\tau \mid \tau \in T\} \cup \{\Xi_m\}, x_0, X_m)$$

similairement à celui de l'équation 2.5.

Toute relation sur L peut donc être induite sur $\mathbf{T_{ts}}$ de la façon suivante. Pour une relation binaire ρ sur L, on *induit* ρ_X une relation binaire sur X par

$$(x, x') \in \rho_X \iff (f^{-1}(x), f^{-1}(x')) \in \rho$$

pour $x, x' \in X$. Ainsi $\ker\,\theta$ (ou toute autre relation d'équivalence sur L) peut être induite sur X de même que le pré-ordre entre préfixes sur les éléments de L, c'est-à-dire pour $x, x' \in X$, $x \leq x'$ si et seulement si il existe $s \in \Sigma^*$ pour laquelle $\xi(x, s) = x'$.

Une projection causale $\theta : L \to M$ pour $L = \overline{L_m} \subseteq \Sigma$ est dotée de la propriété d'observateur si et seulement si $\ker\,\theta$ est une quasi-congruence sur $\widetilde{\mathbf{T}}$. Dans ce cas, la relation d'équivalence π^* correspondante une fois induite sur X peut s'exprimer en termes de Ξ_τ et Ξ_m comme suit :

$$\pi^* = sup\left\{\pi \in \mathcal{E}(X) \mid \pi \leq (\ker\,\theta) \wedge (\pi \circ pre) \wedge \left(\bigwedge_{\tau \in T} (\pi \circ \Xi_\tau)\right) \wedge (\pi \circ \Xi_m)\right\}.$$

On peut contraster le contenu de la figure 2.3 avec le contenu de la figure 7 dans [58] et l'argumentation associée. L'ensemble X_θ dans [58] correspond donc à l'ensemble X dans [60]. Alors $\mathbf{T}$ constitue effectivement un modèle de la paire (L, θ) et le graphe de $\widehat{\mathbf{T}}$

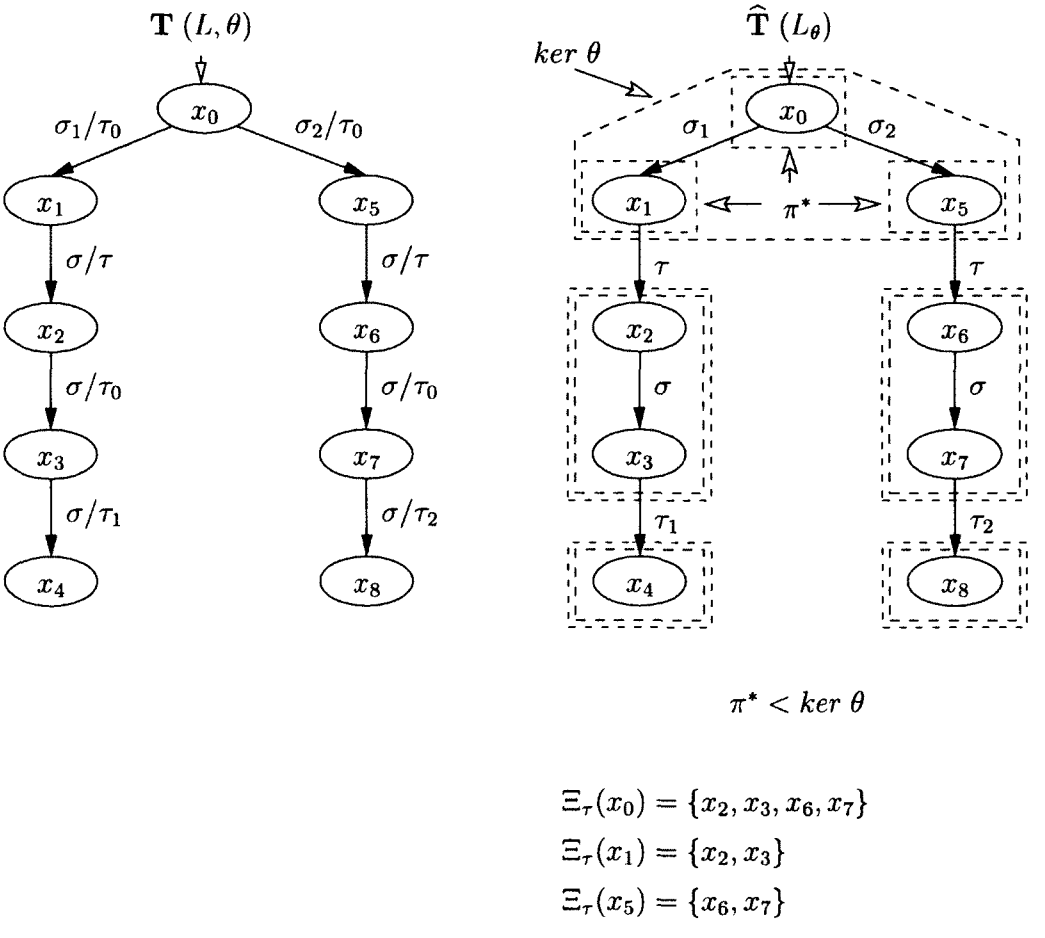


FIGURE 2.3 – Modèles de la paire (L, θ)

n'est rien d'autre qu'un arbre d'accessibilité au même titre que L_θ dans [58]. De plus, en utilisant la projection naturelle $P_T : (\Sigma \cup T)^* \to T$ qui efface toute occurrence de $\sigma \in \Sigma$ dans $L(\widehat{\mathbf{T}})$, on se rend compte que l'ensemble des relations de transition δ_τ pour $\tau \in T$ dans [58] correspondent exactement aux relations Ξ_τ de $\widetilde{\mathbf{T}}$ définies précédemment. On peut donc assimiler (à quelques détails près) la congruence π^* à la congruence observationnelle π_θ définie dans [61] et mentionnée dans [58].

Concernant les preuves de l'existence de π^* le lecteur est invité à consulter [60, 62, 61]. D'après Wonham [62], pour un système dynamique quelconque $\mathcal{S} := (X, \{\alpha_i \mid i \in I\})$, où $i \in I$ est un ensemble fini d'index, $\perp \in \mathcal{E}(X)$ est une quasi-congruence pour $\mathcal{S}$ et l'ensemble des quasi-congruences pour $\mathcal{S}$ forme un sous-treillis supérieur de $\mathcal{E}(X)$ muni de la disjonction. Aussi d'après Wonham [61], pour un ensemble Y et une fonction $\gamma : X \to Y$,

$$\omega^* := sup \left\{ \omega \in \mathcal{E}(X) \mid \omega \leq (\ker \gamma) \wedge \bigwedge_{i \in I} (\omega \circ \alpha_i) \right\}$$

existe et, sous certaines conditions (les fonctions α_i doivent toutes posséder une image finie), ω^* peut être calculée (voir [60]).

On observe finalement dans la figure 2.3, que pour $x \equiv x'(mod\ \pi^*)$, non seulement les chaînes correspondantes $s = f^{-1}(x)$ et $s' = f^{-1}(x')$ de L ont le même *historique abstrait* (c'est-à-dire dans M) mais elles ont aussi le même *développement abstrait* dans M. On note aussi que pour une projection causale $\theta : L \to M$ arbitraire, $\ker\ \theta$ est généralement strictement plus grossière que π^*. Ainsi doter θ de la propriété d'observateur requiert une raffinement de $\ker\ \theta$.

2.3.3.5 Propriété d'observateur et quasi-congruences

Pour un langage H (potentiellement non fermé) sur un alphabet Σ et une chaîne $s \in \Sigma^*$, la fermeture postfixée de s dans H, notée $pos_H(s)$ est constituée de toutes les extensions de s dans H (c'est-à-dire $pos_H(s) := \{u \in H \mid s \leq u\}$). Ainsi pour H fixé, pos_H est une fonction de Σ^* vers $\mathcal{P}(H)$. Posons $H \subseteq \overline{L}$ et $\theta : \overline{L} \to T^*$ causale. On dit que θ est un H-observateur si $\ker\ \theta$ est une quasi-congruence pour le système dynamique $(\overline{L}, pos_H)$.

Ainsi pour un H-observateur θ, si $s \equiv s'(mod\ \ker\ \theta)$, c'est-à-dire $\theta(s) = \theta(s')$, alors $s \equiv s'(mod\ (\ker\ \theta) \circ pos_H)$, c'est-à-dire que pour tout $u \in pos_H(s)$ il existe $u' \in pos_H(s')$

pour lesquels $u \equiv u'(mod \ \ker \theta)$ et vice versa. En d'autres termes, si $\theta(s) = \theta(s')$, alors pour une extension $sv \in H$ de s, il existe une extension $s'v' \in H$ de s' telle que $\theta(sv) = \theta(s'v')$. Ainsi lorsque θ est un H-observateur, si $\theta(s)t \in \theta(H)$ alors peu importe la progression de l'agent dans sa structure de transition, tant et aussi longtemps que son image par θ est égale à $\theta(s)$ l'agent reste toujours en mesure de générer une chaîne de H dont l'image est $\theta(s)t$.

Lemme 2.1. *[60] Soit $\theta : \overline{L} \to T^*$ causale et $H \subseteq \overline{L}$. Alors θ est un H-observateur si et seulement si*

$$(\forall\, s \in \overline{L})(\forall\, t \in T^*)\ \theta(s)t \in \theta(H) \implies (\exists\, u \in \Sigma^*)su \in H \text{ et } \theta(su) = \theta(s)t.$$

Lorsque $H = \overline{L}$ on dit que θ est dotée de la propriété d'observateur, ou simplement que θ est un observateur. Dans ce dernier cas, la formulation suivante du résultat du Lemme 2.1 (en provenance de [58]), plus simple, est possible et souvent utilisée dans les démonstrations. Pour L fermé et $M := \theta(L) \subseteq T^*$, la projection causale $\theta : L \to M$ est un observateur si et seulement si

$$(\forall\, s \in L)(\forall\, \tau \in T)\ \theta(s)\tau \in M \implies (\exists\, u \in \Sigma^+)su \in L \text{ et } \theta(su) = \theta(s)\tau\,.$$

La proposition 8 de Wong et Wonham [58] démontre la propriété suivante des projections causales dotées de la propriété d'observateur :

$$\theta \text{ est un observateur} \iff \theta^{-1} \circ pre_M = pre_L \circ \theta^{-1}$$

où pre_M et pre_L sont respectivement les opérateurs de fermeture préfixée sur M et L. Ainsi pour L et M fermés, $\theta : L \to M$ causale dotée de la propriété d'observateur, on a que pour tout $N \subseteq M$ alors $\theta^{-1}(\overline{N}) = \overline{\theta^{-1}(N)}$.

Il est aussi intéressant de noter qu'un L_m-observateur est aussi un L-observateur pour $L = \overline{L_m}$, d'après la proposition 4.4 dans [14].

2.3.4 Contrôle hiérarchique non bloquant

La théorie du contrôle hiérarchique non bloquant suppose que pour la structure de contrôle de bas niveau $\mathcal{C}_{lo}$ la propriété suivante est toujours vérifiée,

$$(\forall K \in \mathcal{P}(H))\ \overline{K} \in \mathcal{C}_{lo}(H) \implies K \in \mathcal{C}_{lo}(H) \tag{2.6}$$

pour $H \in \mathcal{F}_L$. Cette règle stipule simplement que la contrôlabilité d'un langage est déterminée par la contrôlabilité de sa fermeture préfixée. Cette propriété est toujours vérifiée pour les structures de contrôle standard (voir [55]). On suppose aussi une forme de consistance de contrôle, bien que, selon [58], il suffit que $\mathcal{C}_{hi}(M) = \theta(\mathcal{C}_{lo}(L))$.

On se rappelle que pour $L \in \Sigma^*$ fermé, $L_m \subseteq L$ constitue le langage marqué du système. On dit que $E \subseteq L$ est non bloquant si $\overline{E} = \overline{E \cap L_m}$. On note que si $E = \overline{E} \cap L_m$, c'est-à-dire que E est L_m-fermé, alors E est sûrement non bloquant, mais il n'est pas nécessaire que E soit L_m-fermé pour qu'il soit non bloquant pourvu que $E \subseteq \overline{E \cap L_m}$.

On suppose donc le contexte du contrôle hiérarchique de la figure 2.1 pour le reste de cette section, notamment :

1. $L \in \Sigma^*$ est un langage fermé sur un alphabet fini Σ modélisant un système de bas niveau appelé l'agent, muni d'une structure de contrôle $\mathcal{C}_{lo}$ répondant au critère de l'équation 2.6 ;
2. $M \in T^*$ est un langage fermé sur un alphabet fini T modélisant un système de haut niveau appelé l'abstraction, muni d'une structure de contrôle $\mathcal{C}_{hi}$ que l'on peut supposer être standard ;
3. $\theta : L \to M$ est une projection causale.

Bien sûr on a aussi $L_m \subseteq L$ et $M_m \subseteq M$ définissant respectivement les langages marqués de L et M, et on suppose que $\mathcal{C}_{hi}(M) = \theta(\mathcal{C}_{lo}(L))$.

D'après la proposition 15 dans [58],

$$(\forall E \subseteq M)\ \kappa_L \circ \theta^{-1}(E) \text{ non bloquant} \implies \kappa_M(E) \text{ non bloquant}$$

c'est-à-dire que s'il est possible d'obtenir une synthèse non bloquante dans l'agent à partir de la préimage de l'objectif de contrôle abstrait ($\theta^{-1}(E)$), alors par consistance du contrôle,

il est certain que la synthèse de l'objectif de contrôle dans l'abstraction est elle aussi non bloquante, c'est-à-dire

$$\overline{\kappa_M(E)} = \overline{\overline{\kappa_M(E)} \cap M_m}\,.$$

Le problème du contrôle hiérarchique non bloquant est donc de pouvoir garantir qu'une synthèse d'un objectif de contrôle abstrait ($\kappa_M(E)$ pour $E \subseteq M$), non bloquante dans le système de haut niveau, résulte toujours en une implémentation non bloquante dans l'agent.

Wong et Wonham [58] procèdent à démontrer qu'il suffit que la paire (L, θ) soit *fortement observable* par rapport à un certain sous-ensemble clef de sous-langages de $\overline{\mathcal{C}}_{lo}(L)$ pour assurer le résultat, pourvu que l'on ait $\theta^{-1}(\theta(L_m)) = L_m$, la consistance du marquage.

Premièrement, la paire (L, θ) est dite *fortement observable* si $\theta_{|K}$ (la restriction de θ à K) est dotée de la propriété d'observateur pour tout $K \in \overline{\mathcal{C}}_{lo}(L)$. Deuxièmement on pose

$$\mathcal{X} := \{K \in \overline{\mathcal{C}}_{lo}(L) \mid \kappa_L \circ \theta^{-1}(\theta(K)) = K\}$$

l'ensemble des sous-langages contrôlables et fermés de L qui sont maximaux par rapport à leur projection dans M. On note qu'il peut exister $K_1, K_2 \in \overline{\mathcal{C}}_{lo}(L)$ où $K_1 \subset K_2$, $K_2 \in \mathcal{X}$ et $\theta(K_1) = \theta(K_2)$, c'est le cas dans l'exemple de la figure 2.2 pour $\overline{K'}$. De tels langages ne sont pas *maximaux* par rapport à leur projection dans M et donc ne sont pas inclus dans $\mathcal{X}$. On dit que la paire (L, θ) est *fortement observable par rapport à* $\mathcal{X}$ lorsque $\theta_{|K}$ est un observateur pour tout $K \in \mathcal{X}$.

Il est intéressant de noter, comme mentionné en préambule (exemple de la figure 2.2), que le contrôle hiérarchique est une stratégie indirecte visant à implémenter sur l'agent un comportement contrôlé dont la projection *simule* un comportement contrôlé de l'abstraction générant $\overline{\kappa_M(E)}$. Cette implémentation s'effectue par *calcul* de la loi de contrôle de l'agent V_Σ directement à partir de la loi de contrôle obtenue dans l'abstraction V_T pour la synthèse $\kappa_M(E)$ et correspond (voir [59], postulat à la proposition 12) à $\kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)})$. On vise donc à implémenter $\overline{\kappa_M(E)} = L(V_T/\mathbf{G}_{hi})$ par projection de $\kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)})$ à travers son implémentation $L(V_\Sigma/\mathbf{G}_{lo})$. Ainsi on réfère souvent à $\kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)})$ en tant qu'*implémentation sur l'agent du comportement de la synthèse de haut niveau.* Or en présence de la consistance du contrôle, sous provision que la restriction à l'équation 2.6

soit vérifiée, on a toujours que

$$\begin{aligned} \overline{\kappa_L \circ \theta^{-1}(E)} &\subseteq \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) \\ \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) &\in \mathcal{X} \end{aligned}$$

pour $E \subseteq M$ (voir [58], démonstration de la proposition 16). Dans ces conditions, la proposition 16 de [58] démontre que lorsque la paire (L, θ) est fortement observable par rapport à $\mathcal{X}$ on a en fait que

$$\kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) = \overline{\kappa_L \circ \theta^{-1}(E)}.$$

Par suite on obtient le résultat du théorème 4 [58], à savoir que lorsque $L_m = \theta^{-1}(M_m)$ et que (L, θ) est fortement observable par rapport à $\mathcal{X}$ alors

$$(\forall E \subseteq M)\, \kappa_M(E) \text{ non bloquant} \iff \kappa_L \circ \theta^{-1}(E) \text{ non bloquant}$$

qui est le résultat recherché.

Cependant il faut encore pouvoir assurer que (L, θ) est fortement observable par rapport à $\mathcal{X}$. En proposition 17, Wong et Wonham [58] prouvent que lorsque $\mathcal{C}_{hi}(M) = \theta(\mathcal{C}_{lo}(L))$, pourvu que θ soit dotée de la propriété d'observateur, il suffit qu'il y ait *absence de couplage de contrôle* pour conclure que la paire (L, θ) est fortement observable par rapport à $\mathcal{X}$.

On appelle *absence de couplage de contrôle* ou *absence de partenaire de contrôle* (de l'anglais «*generalized partner-freedom condition*») la propriété suivante :

$$\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M .$$

Un couplage de contrôle se produit *dans l'agent* lorsque la génération de deux ou plusieurs chaînes $t, t' \in M$ est liée à un seul élément de contrôle dans l'agent. Dans cette situation, la suppression (par synthèse abstraite) de l'une de ces chaînes entraîne nécessairement la suppression implicite des autres puisque l'agent ne dispose pas de la finesse de contrôle requise pour les supprimer indépendamment les unes des autres (voir la section B.2).

Ainsi lorsque θ est un L-observateur, l'absence de couplage de contrôle ($\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$) assure que la propriété d'observateur de θ se propage à tous les langages $K \in \mathcal{X}$.

Sommairement, on obtient alors le résultat suivant.

Théorème 6. *[58]*
En présence de la consistance de contrôle, en supposant que la restriction de l'équation 2.6 est vérifiée et que

$\theta^{-1}(M_m) = L_m$;

θ *est dotée de la propriété d'observateur ;*

$\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$,

alors

$$(\forall\, E \subseteq M)\ \kappa_M(E) \textit{ non bloquant} \iff \kappa_L \circ \theta^{-1}(E) \textit{ non bloquant.}$$

On trouve en annexe A une adaptation des propositions et preuves de Wong et Wonham [58] concernant le contrôle hiérarchique non bloquant, pour le cas où on ne dispose que de la forme restreinte de la consistance du contrôle

$$C_{hi}(N) \subseteq \theta(C_{lo}(H)) \iff \theta \circ \kappa_H \circ \theta^{-1}|_{C_{hi}(N)} = id_{C_{hi}(N)}$$

pour $N \in \mathcal{F}_M$ et $H := \theta^{-1}(N) \in \mathcal{F}_L$ (conditions du théorème 2 dans [58]).

2.3.5 Implémentation d'un contrôle hiérarchique non bloquant

L'ensemble des notions couvertes dans cette section sont détaillées au niveau formel dans les sections 6.3 et 6.4 de [55].

Dans cette section, pour alléger l'écriture, on utilise $M(V_N)$ pour signifier $L(V_N/\mathbf{G}_{hi})$ le comportement de $\mathbf{G}_{hi}$ (l'abstraction) sous l'effet d'une loi de contrôle $V_N : M \to \mathcal{P}(T)$ (voir la section 2.3.1.4). Ainsi $M_m(V_N)$ signifie le langage marqué associé à $L(V_N/\mathbf{G}_{hi})$ donc $L(V_N/\mathbf{G}_{hi}) \cap M_m$. Similairement on utilise respectivement $L(V_K)$ et $L_m(V_K)$ à la place de $L(V_K/\mathbf{G}_{lo})$ et $L(V_K/\mathbf{G}_{lo}) \cap L_m$, pour le comportement de $\mathbf{G}_{lo}$ (l'agent) sous le contrôle de $V_K : L \to \mathcal{P}(\Sigma)$ et son langage marqué sous contrôle.

Supposons (L, L_m) et (M, M_m), les modèles d'un *agent* et de son *abstraction* respective, munis respectivement de $\mathcal{C}_{lo}$ et $\mathcal{C}_{hi}$ deux structures de contrôle elles-mêmes munies respectivement de technologies de contrôle correspondantes $\Sigma_c(\cdot)$ et $T_c(\cdot)$. On suppose bien sûr qu'une forme de consistance de contrôle existe entre $\mathcal{C}_{hi}$ et $\mathcal{C}_{lo}$. Supposons en plus que $\theta : L \to M$ est dotée de la propriété d'observateur, qu'elle est libre de couplage de contrôle, c'est-à-dire que $\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$, et qu'aussi le système possède la consistance du marquage $\theta^{-1}(M_m) = L_m$.

Soit maintenant un objectif de contrôle M_m-fermé $E \subseteq M$ et posons $N := \kappa_M(E)$. Supposons N non vide. Alors $N = \kappa_M(E) \subseteq E = \overline{E} \cap M_m \subseteq M_m$ et $N \subseteq M_m$ est M_m-fermé. Par la proposition 85 de Wong [55], une loi de contrôle $V_N : M \to \mathcal{P}(T)$ telle que

$$\begin{aligned} M_m(V_N) &= M_m \cap M(V_N) = \kappa_M(E) = N \\ M(V_N) &= \overline{\kappa_M(E)} = \overline{N} \end{aligned}$$

existe, et cette loi de contrôle est non bloquante.

Posons maintenant $K := \kappa_L \circ \theta^{-1}(E)$. Alors K est non vide et $\theta(K) = N$. Par le théorème 6 de Wong Wonham [58], K est non bloquant. D'autre part puisque $E \subseteq M_m$ par consistance du marquage, $K = \kappa_L \circ \theta^{-1}(E) \subseteq \theta^{-1}(E) \subseteq \theta^{-1}(M_m) = L_m$ et ainsi $K \subseteq L_m$ est L_m-fermé. On a donc encore une fois (par la proposition 85 de Wong [55]) que $V_K : L \to \mathcal{P}(\Sigma)$ telle que

$$\begin{aligned} L_m(V_K) &= L_m \cap L(V_K) = \kappa_L \circ \theta^{-1}(E) = K \\ L(V_K) &= \overline{\kappa_L \circ \theta^{-1}(E)} = \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) = \overline{K} \end{aligned}$$

existe et est non bloquante.

D'après Wong [55], pour que l'on puisse calculer une loi de contrôle sur un agent à partir d'une loi de contrôle sur l'abstraction qui lui correspond, ces deux lois de contrôle doivent être *synchrones*. Or d'après la proposition 86 de Wong [55], en présence de consistance du contrôle, lorsque V_K et V_N existent il suffit que θ soit dotée de la propriété d'observateur et qu'il y ait absence de couplage de contrôle ($\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$) pour que V_K et V_N soient synchrones.

Le calcul s'effectue alors d'après une règle qui dépend des propriétés du système. Par exemple pour le contexte de Zhong et Wonham [66], où la propriété OCC (voir [59], appendice A) procure la consistance de contrôle entre l'abstraction et l'agent, on retrouve en appendice C de [59] une formulation synthétique de la règle à utiliser. Dans ce contexte la loi de contrôle $V_K : L \to \mathcal{P}(\Sigma)$ est définie par la règle suivante :

$$V_K(s) = \Sigma_u \cup \{\sigma \in \Sigma_c \mid (\forall\, s' \in \Sigma_u^*)(\forall\, \tau \in T)\ \theta(s\sigma s') = \theta(s)\tau \implies \tau \in V_N(\theta(s))\} \quad (2.7)$$

pour $s \in L$. La proposition 12 de Wong et Wonham [59] prouve que cette loi de contrôle procure effectivement le résultat désiré à savoir que

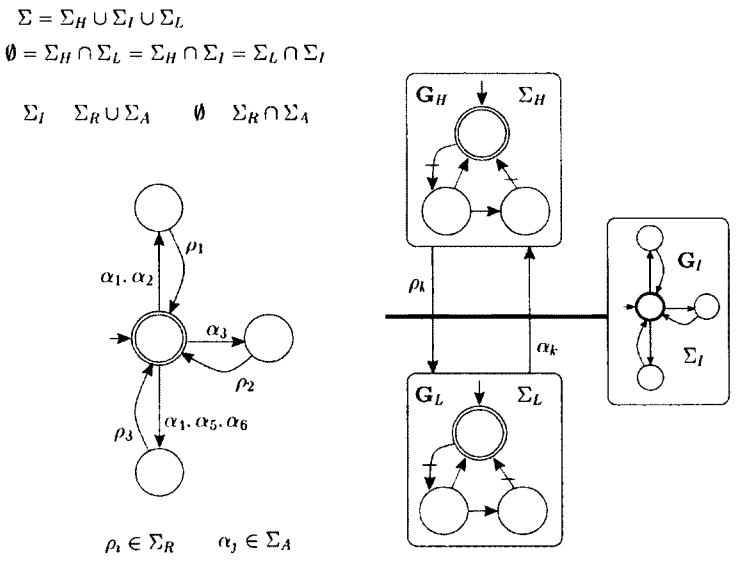
$$\begin{aligned} L_m(V_K) &= L_m \cap L(V_K) = \kappa_L \circ \theta^{-1}(E) = K, \\ L(V_K) &= \overline{\kappa_L \circ \theta^{-1}(E)} = \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) = \overline{K}. \end{aligned}$$

On peut trouver dans la section B.4 plus de détails sur cette construction et sur les concepts reliés.

2.4 Contrôle hiérarchique par interface

Le contrôle supervisé hiérarchique par interface (HISC – *Hierarchical Interface-based Supervisory Control* [35, 38]) a vu le jour sous la forme d'une approche de vérification. On y a plus tard associé la synthèse (voir [36, 9, 52]). Son intérêt réside dans le potentiel qu'il offre (du moins théoriquement) pour la modularisation verticale de composantes et pour la réduction de complexité qu'il apporte au processus de vérification des propriétés de contrôlabilité et d'absence d'impasse. On peut voir HISC comme une approche adaptable aux principes du génie logiciel qui utilise une structure de contrôle conforme à SCT pour permettre la vérification formelle des composantes. La figure 2.4 donne une représentation schématique des principaux éléments de HISC et de leur arrangement conceptuel.

Conceptuellement, il s'agit d'une architecture en deux couches où un système client (composante de haut niveau) utilise les services d'un système serveur (composante de bas niveau) à travers une interface de type commande/réponse. Comme on peut le voir sur la figure, les alphabets d'événements du client, du serveur et de l'interface sont disjoints.



a) Interface, structure en étoile *b) Structure en deux couches*

FIGURE 2.4 – Éléments architecturaux de HISC

Les spécifications de comportement sont données sous forme d'AFD. Le cas illustré dans la figure 2.4 est le plus simple et il ne sert qu'à bien comprendre l'exposition, car il est ensuite étendu à un cas plus général où le client coordonne l'activité de plusieurs serveurs fonctionnant en parallèle, mais dont les alphabets sont tous disjoints deux à deux.

À remarquer que le langage de l'interface $L(\mathbf{G}_I)$ est toujours formé de paires commande/réponse alternées. De plus, une tâche, dans l'interface, ne peut être considérée complète que lorsque la réponse correspondant à une commande donnée au préalable a été reçue. Ainsi, $L_m(\mathbf{G}_I)$ n'est formé que de paires commande/réponse aussi. La forme en étoile illustrée dans la figure n'est que la plus simple des structures possibles. Elle peut être généralisée pour inclure une notion de mode de fonctionnement.

En reformulant légèrement le problème pour inclure l'usage de l'interface et pour exprimer le comportement contrôlé des systèmes, on peut considérer que le comportement contrôlé du système de haut niveau s'exprime par $\mathbf{G}_H := \mathbf{G}_H^p \parallel \mathbf{S}_H$ où $\mathbf{G}_H^p$ exprime le comportement libre du système de haut niveau. La composante $\mathbf{S}_H$ est alors un AFD qui exprime à la fois le contrôle de $\mathbf{G}_H$ et l'usage anticipé de l'interface par le client ($\mathbf{S}_H$ étant défini sur $\Sigma_H \cup \Sigma_I$). On peut faire la même chose pour la paire $\mathbf{G}_L$, $\mathbf{S}_L$, avec $\mathbf{S}_L$ et $\mathbf{G}_L^p$ défini sur $\Sigma_L \cup \Sigma_I$. Alors HISC démontre que, sous certaines conditions (à vérifier à l'aide d'algorithmes donnés dans [38]), le système est contrôlable et non bloquant.

1. Si $\mathbf{G}_H$, $\mathbf{G}_L$ et $\mathbf{G}_I$ sont *serial level-wise nonblocking* (non bloquants *par niveau*) et *serial interface consistent* (consistants à l'interface), le système global est non bloquant.
2. Si $\mathbf{G}_H^p$ et $\mathbf{G}_L^p$, munis respectivement des superviseurs $\mathbf{S}_H$ et $\mathbf{S}_L$ ainsi que de l'interface $\mathbf{G}_I$, sont *serial level-wise controllable* (contrôlables *par niveau*), alors le système global est contrôlable.

Les vérifications se font indépendamment entre deux composantes de niveaux différents (*level-wise* sur $\mathbf{G}_H$ et $\mathbf{G}_L$), procurant un gain appréciable en complexité lorsque le niveau inférieur contient plus d'une composante. La théorie est étendue au cas de plusieurs serveurs (composantes de bas niveau) en donnant un principe permettant la séparation de n systèmes parallèles en n systèmes client/serveur indépendants et équivalents, globalement, au système original (*serial system extraction*).

Des algorithmes sont donnés dans [35, 38] et dans [37], pour l'ensemble des vérifications. Les algorithmes correspondants pour la synthèse sont donnés dans [9].

HISC possède de toute évidence des caractéristiques intéressantes pour la conception modulaire. Mais ses auteurs concèdent qu'elle présente tout de même certains problèmes.

1. La méthode ne garantit pas un comportement contrôlable et non bloquant qui soit maximal au sens de la théorie originale ; un inconvénient mineur à concéder pour obtenir la modularité.
2. La méthode semble confiner à une approche client/serveur en deux couches [37]. Du moins les auteurs n'abordent pas les difficultés d'extension à plus de deux couches. Il est probable qu'une extension vers le haut (d'un système parallèle) requiert de reprendre la vérification de chaque module globalement. Cependant les gains en termes de réduction de complexité demeurent valables.

On peut noter aussi que lorsque l'on se limite à utiliser cette variante dans un rôle de vérification, la spécification du système de haut niveau (système maître) doit être donnée sous forme d'un AFD monolithique (le superviseur $\mathbf{S}_H$). Ceci peut souvent se révéler être un exercice difficile, et dont on ne peut pas aisément recouvrer l'ensemble des objectifs de contrôle initiaux.

2.5 Approche états ou modèle avec prédicats

Il est souvent plus facile d'utiliser un formalisme de logique pour exprimer des propriétés relatives au contrôle sur les systèmes. À cet effet, Wonham, Rogers et Ma [62, 39] formulent une approche duale à l'approche linguistique de SCT utilisant des prédicats comme base pour la spécification des éléments du contrôle.

Cette approche utilise des prédicats qui expriment des conditions sur les états du système $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$. Tout prédicat $P : Q \to \{0, 1\}$ peut toujours être identifié à un élément de 2^Q par $Q_P := \{q \in Q \mid P(q) = 1\}$. On dit que $q \models P$ (q satisfait P) si et seulement si $q \in Q_P$. Si Q est fini, l'ensemble des prédicats sur Q, dénoté $Pred(Q)$, est fini et identifié à 2^Q en ce sens que P agit comme une indicatrice sur Q_P.

En plus des opérateurs logiques usuels (conjonction, disjonction et négation), on définit

aussi la relation $\preceq$, où $P_1 \preceq P_2$ si et seulement si $P_1 \wedge P_2 = P_1$, c'est-à-dire $P_1 \Rightarrow P_2$. Considérant que $\preceq$ dans $Pred(Q)$ est analogue à $\subseteq$ dans 2^Q, il est clair que $(Pred(Q), \preceq)$ définit un treillis complet dont la borne supérieure, $\sup(Pred(Q)) = \mathit{true}$ (le prédicat le plus faible), correspond à Q, et la borne inférieure, $\inf(Pred(Q)) = \mathit{false}$ (le prédicat le plus fort, au sens du plus restrictif), correspond à $\emptyset$.

Dans cet espace on définit $R(\mathbf{G}, P)$, pour $P \in Pred(Q)$, le prédicat d'accessibilité correspondant à $\mathbf{G}$ sous P par

1. $q_0 \models P \Rightarrow q_0 \models R(\mathbf{G}, P)$;
2. $q \models R(\mathbf{G}, P) \wedge \delta(q, \sigma)! \wedge \delta(q, \sigma) \models P \Rightarrow \delta(q, \sigma) \models R(\mathbf{G}, P)$;
3. aucun autre état ne satisfait $R(G, P)$.

$R(\mathbf{G}, P)$ est identifié à l'ensemble des états que l'on peut atteindre dans $\mathbf{G}$ à partir de q_0, en n'empruntant que des trajectoires d'états satisfaisant P. Clairement, pour un générateur $\mathbf{G}$ fixé, $R(\mathbf{G}, \cdot) : Pred(Q) \rightarrow Pred(Q)$ définit un foncteur.

Pour exprimer la condition de contrôlabilité dans cet espace il est nécessaire de définir un autre foncteur $M_\sigma : Pred(Q) \rightarrow Pred(Q)$, où $q \models M_\sigma(P) \Leftrightarrow \delta(q, \sigma) \models P$ lorsque $\delta(q, \sigma)!$. Alors, $P \in Pred(Q)$ est dit contrôlable par rapport à $\mathbf{G}$ lorsque

$$P \preceq R(\mathbf{G}, P) \wedge (\forall \sigma \mid \sigma \in \Sigma_u : P \preceq M_\sigma(P)).$$

Il est clair que $R(\mathbf{G}, P) \preceq P$ en général, donc la contrôlabilité implique que $R(\mathbf{G}, P) = P$. Notamment, P est accessible dans $\mathbf{G}$ et invariant sous le flot des événements incontrôlables. Le prédicat *false* est contrôlable par convention.

Dans cet espace, pour établir le contexte du contrôle, on ne s'intéresse qu'à des lois de contrôle où l'état courant du système $\mathbf{G}$ suffit pour déterminer la rétroaction. De telles lois de contrôle sont désignées par l'appellation SFBC (de l'anglais *State FeedBack Control policy*). Formellement, une SFBC f est une fonction totale $f : Q \rightarrow \mathcal{C}$, où

$$\mathcal{C} := \{\Sigma' \subseteq \Sigma \mid \Sigma_u \subseteq \Sigma'\}.$$

Alors $f(q)$ spécifie l'ensemble des événements autorisés à l'état $q \in Q$, qui contient toujours tout l'ensemble des événements incontrôlables Σ_u ; tous les autres événements sont

ainsi inhibés. Une SFBC f définit implicitement une famille $\{f_\sigma \mid \sigma \in \Sigma\}$ de prédicats $f_\sigma : Q \to \{0,1\}$, où $f_\sigma(q) = 1$ si et seulement si $\sigma \in f(q)$. Le comportement en boucle fermée $\mathbf{G}^f := (Q, \Sigma, \delta^f, q_0, Q_m)$ du système sous l'action de f est alors défini par

$$\delta^f(q,\sigma) := \begin{cases} \delta(q,\sigma) & \text{si } \delta(q,\sigma)! \text{ et } f_\sigma(q) = 1 \\ \textit{indéfinie} & \text{autrement.} \end{cases}$$

Il est évident que si f est une SFBC pour $\mathbf{G}$, alors $R(\mathbf{G}^f, P) \preceq R(\mathbf{G}, P)$ dénote le prédicat d'accessibilité de $\mathbf{G}^f$.

La théorie établit qu'un prédicat non trivial $P \in Pred(Q)$ est contrôlable par rapport à $\mathbf{G}$ si et seulement si il existe une SFBC f pour $\mathbf{G}$ telle que $R(\mathbf{G}^f, true) = P$, c'est-à-dire que f synthétise P sur $\mathbf{G}$.

De la même façon que l'on peut définir $R(\mathbf{G}, P)$, on peut définir $CR(\mathbf{G}, P)$ le prédicat de co-accessibilité. Un prédicat P est dit co-accessible si $P \preceq CR(\mathbf{G}, P)$ et non bloquant si $R(\mathbf{G}, P) \preceq CR(\mathbf{G}, P)$. Or $CR(\mathbf{G}, P) \preceq P$ en général, donc si P est contrôlable et non bloquant alors $P = R(\mathbf{G}, P) \preceq CR(G, P) \preceq P$ et ainsi P est co-accessible. Et on obtient que $R(\mathbf{G}^f, true) = P = R(\mathbf{G}, P) = CR(\mathbf{G}, P)$. Sous ces conditions, la loi de contrôle est donnée par une SFBC $f : Q \to C$ définie par l'ensemble des prédicats $f_\sigma = M_\sigma(P)$.

Pour la synthèse, lorsque P n'est pas contrôlable a priori, on considère l'ensemble $\mathcal{CP}(P) := \{K \in Pred(Q) \mid K \preceq P \text{ et } K \text{ est contrôlable}\}$. Cet ensemble est démontré non vide et stable sous la disjonction [62]. Il admet donc une borne supérieure dénotée $\sup\mathcal{CP}(P)$.

De plus, on définit le foncteur $[\cdot] : Pred(Q) \to Pred(Q)$ par

$$q \models [R] \Leftrightarrow (\exists\, q' \mid q' \in Q : q' \models R \wedge (\exists\, u \mid u \in \Sigma_u^* : \delta(q,u)! \wedge \delta(q,u) = q')).$$

Il est évident que $R \preceq [R]$ et $[R]$ dénote le plus grand sous-ensemble d'états de $\mathbf{G}$ pouvant atteindre un état $q' \models R$ par au moins une séquence composée uniquement d'événements incontrôlables dans $\mathbf{G}$. On appelle $[R]$ le champ incontrôlable de R.

On peut démontrer que $\sup\mathcal{CP}(P) = R(\mathbf{G}, \neg[\neg P])$, et bien sûr $\sup\mathcal{CP}(P) \neq false$ si et seulement si $q_0 \models \neg[\neg P]$ (voir [62]). Sous ces conditions, la loi de contrôle prend la

forme d'une SFBC $f : Q \to \mathcal{C}$ définie, pour tout $\sigma \in \Sigma_c$, par

$$f_\sigma(q) := \begin{cases} 1 & \text{si } \delta(q,\sigma)! \wedge \delta(q,\sigma) \models \neg\left[\neg P\right] \\ 0 & \text{autrement.} \end{cases}$$

De façon analogue à ce qui précède, pour la synthèse non bloquante avec P arbitraire, on considère l'ensemble

$$\mathcal{C}^2\mathcal{P}\left(P\right) := \left\{K \in Pred(Q) \mid K \preceq P, K \text{ contrôlable et } K \text{ co-accessible}\right\}.$$

non vide et stable sous la disjonction [62]. Cet ensemble admet une borne supérieure dénotée $\sup\mathcal{C}^2\mathcal{P}\left(P\right)$. On définit le foncteur $\Omega_P\left(\cdot\right)$ par

$$\Omega_P\left(K\right) := P \wedge CR(\mathbf{G}, \sup\mathcal{CP}(K)) = P \wedge CR(\mathbf{G}, R(\mathbf{G}, \neg\left[\neg K\right])).$$

On peut ainsi caractériser $S = \sup\mathcal{C}^2\mathcal{P}\left(P\right)$ comme le plus grand point fixe de $\Omega_P\left(\cdot\right)$, et bien sûr $\sup\mathcal{C}^2\mathcal{P}\left(P\right) \neq \mathit{false}$ si et seulement si $q_0 \models \sup\mathcal{C}^2\mathcal{P}\left(P\right)$. Sous ces conditions, la loi de contrôle prend la forme d'une SFBC $f : Q \to \mathcal{C}$ définie, pour tout $\sigma \in \Sigma_c$, par

$$f_\sigma(q) := \begin{cases} 1 & \text{si } \delta(q,\sigma)! \wedge \delta(q,\sigma) \models \sup\mathcal{C}^2\mathcal{P}\left(P\right) \\ 0 & \text{autrement.} \end{cases}$$

En conclusion, cette approche peut sembler restrictive puisque la rétroaction est donnée à partir de l'état courant du système (c'est-à-dire basée sur l'espace d'états de $\mathbf{G}$). Dans la forme donnée jusqu'à maintenant, elle ne peut pas tenir compte de l'historique de l'activité du système, comme par exemple si une décision de contrôle devait dépendre du nombre de fois que le système a produit une pièce de type «A» nécessitant ainsi l'usage d'un compteur. Cependant dans [62], la théorie formule une extension de la forme de la loi de contrôle incorporant une notion de *mémoire* appelée une DSFBC (pour *dynamic* SFBC). Munie de cette extension cette approche se révèle capable de résoudre les mêmes problèmes que le modèle linguistique original de SCT.

2.6 Méthodes symboliques en synthèse de contrôleurs

Les méthodes dites symboliques jouissent depuis longtemps d'une faveur particulière dans les techniques de vérification. Rappelons qu'en vérification, on tente de prouver qu'un système, dont la structure et la dynamique sont adéquatement modélisées, satisfait une certaine propriété P (habituellement donnée dans un formalisme logique). En termes d'outils de calcul symbolique, les méthodes de vérification couvrent une large gamme allant des prouveurs de théorèmes (par exemple le langage de spécification *B* [1]) au *model-checking* [26]. Une description même sommaire de ce domaine se situe bien au-delà de la portée de cette thèse. Cependant certains outils utilisés pour faire de la vérification de systèmes par *model-checking*, tels *Supremica* [2] et UPPAAL [4], sur des modèles utilisant des structures à transitions similaires à des AFD ont donné lieu à des travaux connexes en synthèse de contrôleurs SCT (voir [64, 63]). Il importe donc de comprendre ce que l'on entend par méthodes symboliques en synthèse de contrôleurs SCT et d'en situer l'importance relative par rapport aux autres travaux.

La locution *méthodes symboliques* en synthèse de contrôleurs SCT réfère à l'usage d'un outil particulier, appelé diagramme de décision (MDD, IDD ou BDD, respectivement *Multi-valued*, *Integer* et *Binary Decision Diagram*), dans le calcul des solutions de synthèse. Cet outil vise à éviter d'avoir à expliciter l'espace des états des modèles de systèmes (le produit G || S) pendant les calculs. En effet cet espace d'états atteint très vite un niveau trop élevé pour qu'il soit réaliste de les manipuler explicitement, même sur un ordinateur moderne. La méthode utilise une représentation dite *symbolique* (par exemple, les prédicats présentés dans la section 2.5) pouvant être encodée de façon plus compacte qu'une énumération exhaustive, par l'intermédiaire de diagrammes de décision.

L'usage de diagrammes de décision n'est cependant pas une solution au problème d'explosion combinatoire des modèles en SCT. La complexité de ces diagrammes peut elle-même devenir ingérable (le diagramme devient trop volumineux rendant impossible la synthèse). Cependant l'expérience, tant en vérification qu'en synthèse de contrôleurs, tend à prouver que l'usage de diagrammes de décision permet la synthèse de systèmes de taille plus réaliste, d'où leur intérêt. La méthode de synthèse doit cependant y être adaptée.

2.6.1 Diagramme de décision

Qu'est-ce qu'un diagramme de décision? Un traitement formel de cette question est donné en [30], ce qui suit n'est qu'une illustration du concept.

On peut observer qu'une fonction finie $f : P_1 \times P_2 \times \ldots \times P_n \to Y = \{0, 1, \ldots, m-1\}$, où $P_i = \{0, 1, \ldots, p_i - 1\}$, peut être décomposée par rapport à n'importe laquelle de ses variables à l'aide de l'identité suivante, nommée *décomposition de Shannon*.

$$f = \sum_{j=0}^{p_i - 1} x_i^j \cdot f_{x_i^j}$$

Cette décomposition utilise l'indicatrice de Lagrange $x_i^j : P_i \to \{0, 1\}$ qui vaut 1 si et seulement si $x_i = j$, et le co-facteur fonctionnel de f en $x_i = j$ défini par

$$f_{x_i^j}(x_1, x_2, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, j, x_{i+1}, \ldots, x_n).$$

Ainsi, une fonction finie quelconque jouit d'une représentation par graphe simple, dont on peut trouver un exemple en figure 2.5. Un tel graphe porte le nom de MDD (de l'anglais pour *Multi-valued Decision Diagram*). Les noeuds internes représentent les variables et sont ordonnés par niveau dans le graphe. Le graphe est acyclique et ses noeuds terminaux représentent les valeurs prises par la fonction. Évaluer la fonction en un point consiste à circuler dans le graphe à partir de la racine jusqu'à l'un de ses noeuds terminaux en empruntant, à chaque noeud interne, la branche correspondant à la valeur de la variable représentée par ce noeud. La navigation n'est donc pas arbitraire, en ce sens que l'évaluation de la fonction doit toujours se faire dans l'ordre des variables implicite au graphe, il est donc dit ordonné (OMDD).

On y constate aussi beaucoup de redondances que l'on peut éliminer à profit sans changer la nature acyclique du graphe ou l'ordre des variables dans le graphe. En particulier, on peut éliminer la redondance parmi les noeuds terminaux. Ce faisant, on constate que les arcs du noeud de gauche pour la variable y pointent tous sur le même noeud terminal, aussi peut-on éliminer ce noeud en redirigeant l'arc $x = 0$ directement vers le noeud terminal représentant $f(0, y) = 0$.

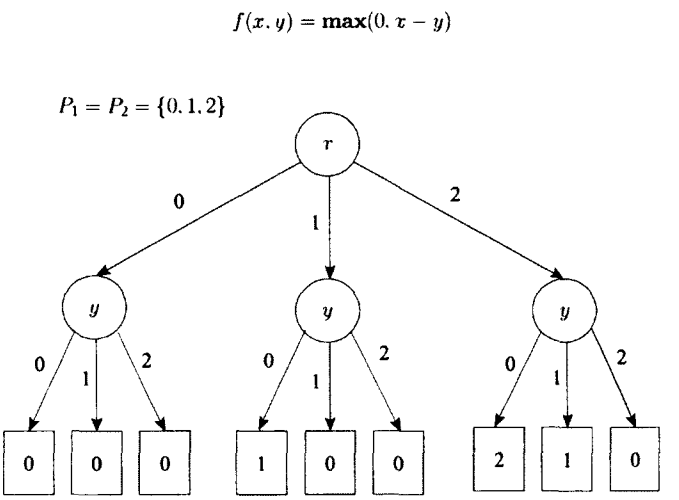


FIGURE 2.5 – Diagramme de décision d'une fonction finie

Plus généralement, on peut appliquer les deux règles suivantes appelées *règles d'élimination de Shannon* en procédant à partir des noeuds terminaux vers la racine.

1. Tout noeud n'ayant qu'un successeur (c'est-à-dire que les arcs dont il est l'origine pointent tous sur le même noeud) peut être éliminé en redirigeant les arcs de ses prédécesseurs vers son successeur.
2. Pour toute paire de noeuds (n_1, n_2) représentant des sous-graphes idempotents (c'est-à-dire qu'ils ont le même nombre d'arcs et chaque paire d'arcs correspondants pointent vers le même successeur), on peut éliminer l'un d'entre-eux, disons n_1, en redirigeant les arcs de ses prédécesseurs vers n_2.

Le résultat de l'application de ces règles donne un graphe réduit qu'on appelle ROMDD (*Reduced Ordered Multi-valued Decision Diagram*). Lorsque le contexte est clair, MDD signifie ROMDD. Cette dernière forme est considérée canonique en ce sens qu'il n'existe qu'un et un seul ROMDD représentant une fonction donnée f (l'ordre des variables est une caractéristique d'un ROMDD). Lorsque l'ordre des variables est optimal, le graphe prend en général beaucoup moins d'espace qu'une représentation exhaustive (par exemple, par

table) de f.

Ces résultats se généralisent facilement aux fonctions booléennes. On appelle alors le graphe un BDD ou ROBDD (*Reduced Ordered Binary Decision Diagram*). Gunnarsson (voir [26]) utilise le nom IDD (*Integer Decision Diagram*) dans le cas de fonctions finies à valeur dans $\{0, 1\}$, particulièrement utiles pour exprimer des fonctions caractéristiques (prédicats d'appartenance) sur les ensembles d'états d'une composition synchrone d'AFD. En effet, pour

$$\mathbf{G} = \|_{i \leq n} \mathbf{G}_i, \text{ un état } x = (x_1, x_2, \ldots, x_n) \in \times_{i \leq n} X_i,$$

où X_i est l'ensemble d'états de la composante $\mathbf{G}_i$ pour n fini, il est aisé de voir qu'un prédicat P tel que défini dans la section 2.5 (une indicatrice sur Q_P) peut être encodé par un IDD.

Moyennant un encodage judicieux [30], les IDD peuvent être encodés à l'aide de BDD. Ainsi IDD et BDD, implémentés à l'aide de structures de données adéquates, permettent la conception d'algorithmes efficaces pour les opérations suivantes [12] :

1. l'évaluation de $f(x)$ pour x donné ;
2. la réduction d'après les règles d'élimination de Shannon ;
3. le test d'équivalence de fonctions $f_1 = f_2$ (par égalité de pointeurs) ;
4. l'énumération et la détermination de la cardinalité de f ;
5. la synthèse $f = f_1 \bullet f_2$, où $\bullet$ représente n'importe quel opérateur booléen binaire ;
6. la négation logique ;
7. la substitution de variable par une constante $f_{|x_i=k}$ ou une autre fonction g définie sur le même domaine $f_{|x_i=g}$;
8. la quantification existentielle et universelle.

Il existe une grande variété de diagrammes de décision et une riche littérature scientifique sur le sujet. Chaque forme de diagramme est ajustée sur mesure à la nature du problème à traiter. Tous cependant souffrent de la même limitation à savoir que la taille des diagrammes (pendant les calculs) dépend de façon critique de l'ordonnancement des variables utilisé. Un mauvais ordonnancement peut rendre la technique inutilisable. Il y

a donc une large part de la littérature consacrée aux heuristiques d'ordonnancement des variables.

2.6.2 Diagramme de décision en synthèse de contrôleurs

Que ce soit en vérification ou en synthèse de contrôleurs, l'usage de diagrammes de décision présente certaines difficultés, même dans le contexte où les systèmes sont modélisés par un produit synchrone.

1. Représenter un ensemble d'états par une indicatrice qu'on peut encoder dans un IDD est une démarche systématique mais peu supportée par la théorie. Le modèle par produit synchrone lui-même ne se prête pas très bien à cette démarche, puisqu'on ne peut pas formuler de loi de projection simple à partir d'un modèle obtenu par produit synchrone.
2. Les deux approches (vérification ou synthèse) requièrent l'évaluation du prédicat représentant la portion accessible (ou co-accessible) de l'espace des états $R(\mathbf{G}, P)$. Or $R(\mathbf{G}, \cdot)$ est un foncteur de $Pred(Q) \rightarrow Pred(Q)$ qui doit transformer une indicatrice (un IDD) en une autre. La théorie des diagrammes de décision ne dit rien a priori sur ce genre de manipulation. De plus, cette transformation doit utiliser la fonction de transition δ qu'il faudra de toute évidence reformuler sous forme de diagramme de décision pour le calcul de foncteurs tels que $R(\mathbf{G}, \cdot)$.

La première difficulté peut être résolue assez facilement en proposant des algorithmes appropriés. Ce sujet est abordé dans [6], mais sans support de la part du modèle lui-même, le fondement théorique ne peut être que chancelant. Cette difficulté est bien résolue dans [39] mais sur un modèle légèrement différent.

La deuxième difficulté est abondamment traitée dans [40, 6] et [30] dans le contexte de la vérification, mais semble faire appel à des notions peu utilisées dans le cadre de SCT.

Les travaux de Zhang et Wonham [63, 64] présentent un algorithme formel pour la synthèse de contrôleurs non bloquants, dérivé de l'approche prédicative, et réalisé à partir de spécifications données sous forme d'AFD. Dans ce cas cependant, la loi de contrôle n'est pas une SFBC, mais une loi de contrôle dérivée de l'espace d'états d'un superviseur

adéquat. Des procédures *ad hoc* sont données pour réaliser $R(\mathbf{G}, P)$ et $CR(\mathbf{G}, P)$ mais sans en donner le fondement théorique.

Les travaux de Ma et Wonham [39] reprennent l'approche prédicative dans le contexte d'un nouveau formalisme de modélisation, les STS (de l'anglais *State Tree Structures*). La modélisation par STS procure le support formel requis pour une résolution élégante des deux problèmes.

2.7 Structures en arborescence d'états

Cette section traite principalement des travaux de Ma et Wonham sur l'utilisation d'un nouvel outil de modélisation, les structures en arborescence d'états (STS – *State Tree Structures*). Puisqu'il s'agit d'un nouveau formalisme, les résultats de SCT ont dû être démontrés à nouveau dans le cadre des STS. Le lecteur peut consulter [39] pour obtenir le traitement détaillé ; cette section n'offre qu'une exposition condensée. Les structures en arborescence d'états ont d'abord été proposées par Wang dans [54] et ensuite formalisées en termes linguistiques par Gohari [24]. Dans sa forme originale, la synthèse de contrôleurs à partir de STS souffrait de certaines limitations qui ont été levées par Ma et Wonham.

Pour plus de clarté, cette section est divisée en trois sous-sections. Il faut d'abord décrire le support formel (les STS) et ses caractéristiques. Ensuite, il faut considérer comment ce support est utilisé pour la synthèse. Finalement, il est intéressant de considérer comment la synthèse elle-même peut être implémentée par l'usage de diagrammes de décision.

2.7.1 L'outil de modélisation STS

Les STS sont un formalisme pour la description d'espaces d'états hiérarchiques inspiré des *statecharts* (automates hiérarchiques) décrits par Harel et Politi dans [27]. Les STS sont formées :

1. d'une structure algébrique pour en décrire l'aspect statique, l'arborescence d'états dénotée ST ;
2. d'une structure algébrique pour en décrire l'aspect dynamique, les *holons*, une structure à transition similaire aux AFD.

Pour une description formelle complète le lecteur peut consulter [39]. Ce qui suit n'est qu'un énoncé des principales propriétés de chaque structure et de leur combinaison finale en une *structure en arborescence d'états*.

2.7.1.1 Structure statique, l'arborescence d'états

La figure 2.6 (extraite de [39]) présente le modèle d'un système sous forme d'un *statechart*. On peut trouver en figure 2.7 une représentation de l'arborescence d'états correspondante. L'arborescence d'états d'un système, dénoté $\mathrm{S}T$, est clairement une structure d'arbre et en hérite toutes les caractéristiques. L'arbre est composé de trois types de noeuds : les super-états ET et les super-états OU, formant les noeuds internes de l'arbre, et les noeuds simples comme feuilles. La structure est définie récursivement de sorte que tout sous-arbre d'une arborescence d'états $\mathrm{S}T$ est lui-même une arborescence d'états. On parle d'arborescence d'états enracinée en M_1, dénotée $\mathrm{S}T^{M_1}$, pour un sous-arbre. Une fonction de typage $\mathcal{T}$ sert à dénoter le type des noeuds ; en figure 2.7, $\mathcal{T}(M_1) = or$, $\mathcal{T}(\textit{Small Factory}) = and$, $\mathcal{T}(I_2) = simple$.

On dénote $\mathcal{E}$ la fonction d'expansion directe d'un noeud retournant l'ensemble des noeuds qui le composent directement, $\mathcal{E}^+$ sa fermeture transitive et $\mathcal{E}^*$ sa fermeture transitive et réflexive.

On dit qu'une arborescence d'états est bien formée (de l'anglais *well-formed*) si tous ses super-états ET ne sont composés que de super-états (OU ou ET). La structure algébrique est dénotée $\mathrm{S}T = (X, x_0, \mathcal{T}, \mathcal{E})$, où X est un ensemble d'états structurés, $x_0 \in X$ un état spécial appelé la racine, $\mathcal{T} : X \rightarrow \{or, and, simple\}$ la fonction de typage et $\mathcal{E} : X \rightarrow 2^X$ la fonction d'expansion directe.

Il est aussi utile de caractériser les relations entre les noeuds d'une $\mathrm{S}T$. On dit que $x \leq y$ (x est un ancêtre de y), si $y \in \mathcal{E}^*(x)$, et que $x < y$ si x est un ancêtre propre de y. Généralement x et y sont affiliés si $x \leq y$ ou $y \leq x$. Cette relation nous permet d'établir la notion de plus proche parent (NCA – *nearest common ancestor*) et d'affirmer qu'entre les noeuds d'une $\mathrm{S}T$, il ne peut exister qu'une et une seule des trois relations suivantes (mutuellement exclusives).

1. Soit x et y sont affiliés.

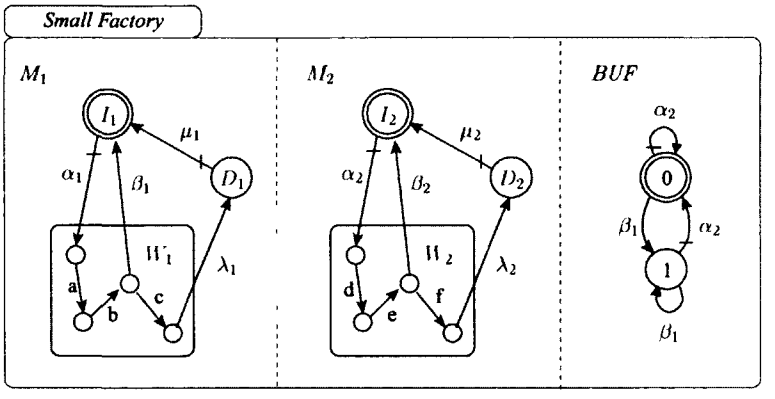


FIGURE 2.6 – Modèle du problème *Small Factory*

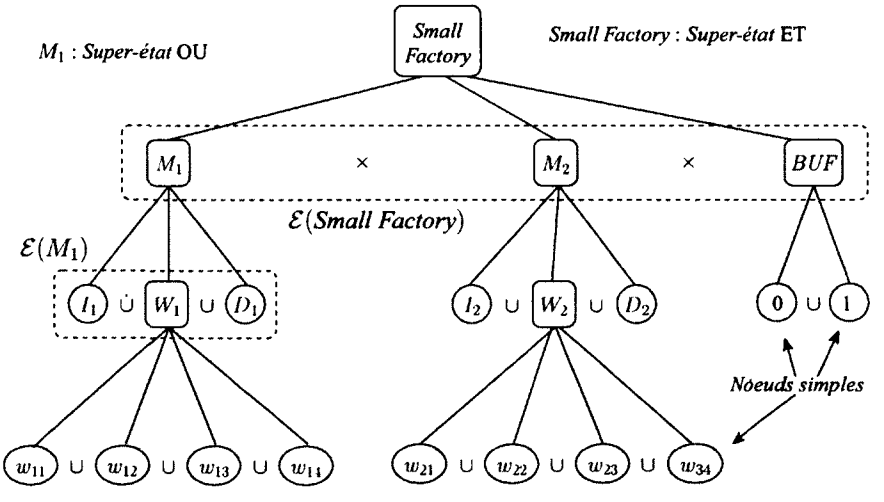


FIGURE 2.7 – Arborescence d'états

2. Soit $x \mid y$, et x est dit concurrent à y, en ce sens que le système peut être dans les deux états simultanément, et leur plus proche parent est un super-état ET.
3. Soit $x \perp y$, et x est en relation d'exclusion mutuelle avec y, en ce sens que le système est soit dans l'un de ces états soit dans l'autre, et leur plus proche parent est un super-état OU.

La théorie développe ensuite la notion de sous-arborescence d'états (*Sub-State Tree*) dénoté $subST$, à distinguer clairement de $\mathrm{S}T^x$ (l'arborescence d'états enracinée en x) qui est un sous-arbre d'une $\mathrm{S}T$.

Une sous-arborescence d'états est toujours relative à une arborescence d'états de référence, en général dénotée $\mathrm{S}T$. Une sous-arborescence d'états $subST$ de $\mathrm{S}T$ est construite à partir de $\mathrm{S}T$ par un processus d'élagage particulier où l'on ne coupe des branches que sur les super-états OU en prenant soin de ne pas toutes les couper. Le résultat est une nouvelle arborescence d'états (bien formée) de racine x_0 (comme $\mathrm{S}T$) et dont tous les chemins partant de la racine aboutissent à une feuille (un noeud simple). Cette structure dénote un sous-ensemble des états du système.

En général, on peut montrer que l'arborescence de référence d'un système $\mathrm{S}T$ dénote de façon compacte l'ensemble des états du même système qu'on aurait obtenu par composition synchrone. On appelle souvent un modèle obtenu par composition synchrone un modèle *plat*, en référence à sa structure horizontale. Dans la figure 2.6, il y a trois composantes (des super-états OU) dans un super-état ET, le produit synchrone est implicite. L'arborescence correspondante (figure 2.7) permet aisément d'énumérer toutes les *configurations* que peut prendre le système (les combinaisons d'états occupés concurremment par l'ensemble des composantes). Cette expression de l'espace d'états est plus compacte qu'une énumération explicite des éléments du produit synchrone. Une $\mathrm{S}T$ agit donc comme un *symbole* pour dénoter un ensemble d'états d'un modèle plat. Une sous-arborescence d'états $subST$ agit comme un sous-ensemble des états de $\mathrm{S}T$, l'arborescence des états d'un système.

À l'instar de l'inclusion chez les ensembles, on peut établir une relation d'inclusion entre arborescences d'états, $\mathrm{S}T_1 \leq \mathrm{S}T_2$. On définit aussi $ST(\mathrm{S}T)$ l'ensemble des sous-arborescences d'états qu'on peut former à partir de $\mathrm{S}T$, où $\emptyset \in ST(\mathrm{S}T)$ dénote la sous-arborescence d'états vide. Alors $ST(\mathrm{S}T)$ est similaire à 2^Q, où Q est l'ensemble des états d'un système. On peut démontrer que $(ST(\mathrm{S}T), \leq)$ est un treillis complet lorsqu'on définit

adéquatement deux opérateurs similaires à la conjonction (le *meet* $\wedge$) et à la disjonction (le *join* $\vee$) d'ensembles.

Enfin, pour pouvoir nous ramener à une notion d'états élémentaires d'un système, la théorie définit la notion d'arborescence d'états élémentaire (*Basic State Tree*). Cette notion joue un rôle analogue à ce que Harel nomme une *configuration* pour les *statecharts* (dans [27]). Un exemple d'arborescence d'états élémentaire est donné en figure 2.8.

On définit donc $\mathcal{B}(\mathrm{S}T)$ l'ensemble des arborescences élémentaires qu'on peut former sur $\mathrm{S}T$. Notez qu'il y a une bijection entre $\mathcal{B}(\mathrm{S}T)$ et l'ensemble des états Q d'un système plat.

Vu la similitude de structure entre $(ST(\mathrm{S}T), \leq)$ et $\left(2^Q, \subseteq\right)$ d'une part, et celle entre Q et $\mathcal{B}(\mathrm{S}T)$ d'autre part, il semble pertinent d'examiner les relations entre les opérations $\wedge$ (*meet*) et l'intersection d'ensembles, et aussi entre $\vee$ (*join*) et l'union d'ensembles.

1. Si $\mathrm{S}T_\wedge = \mathrm{S}T_1 \wedge \mathrm{S}T_2$, alors $\mathcal{B}(\mathrm{S}T_\wedge) = \mathcal{B}(\mathrm{S}T_1) \cap \mathcal{B}(\mathrm{S}T_2)$.
2. Si $\mathrm{S}T_\vee = \mathrm{S}T_1 \vee \mathrm{S}T_2$, alors $\mathcal{B}(\mathrm{S}T_\vee) \supseteq \mathcal{B}(\mathrm{S}T_1) \cup \mathcal{B}(\mathrm{S}T_2)$, avec une inclusion stricte en général.

La propriété (2) empêche effectivement la synthèse dans l'espace des arborescences d'états, il faut donc trouver une autre expression pour les procédures de synthèse avec cet outil de modélisation.

2.7.1.2 Structure dynamique, le *holon*

Les *holons*, dont une illustration schématique est donnée en figure 2.9, sont des structures de transitions similaires à des AFD. Dans une structure en arborescence d'états les *holons* ne sont associés qu'aux super-états OU pour en décrire l'aspect dynamique. Il n'est pas nécessaire d'associer des *holons* aux super-états ET puisque dans une arborescence d'états *bien formée*, les composantes d'un super-état ET sont aussi des super-états.

Intuitivement, un holon est un AFD partitionné, c'est en fait une forme d'automate fini puisqu'il n'est pas nécessaire qu'il soit déterministe ; il peut comporter plusieurs états initiaux. L'ensemble de ses états est partitionné en sous-ensembles d'états externes X_E, et internes X_I. On a alors $X = X_E \cup X_I$ et $X_E \cap X_I = \emptyset$.

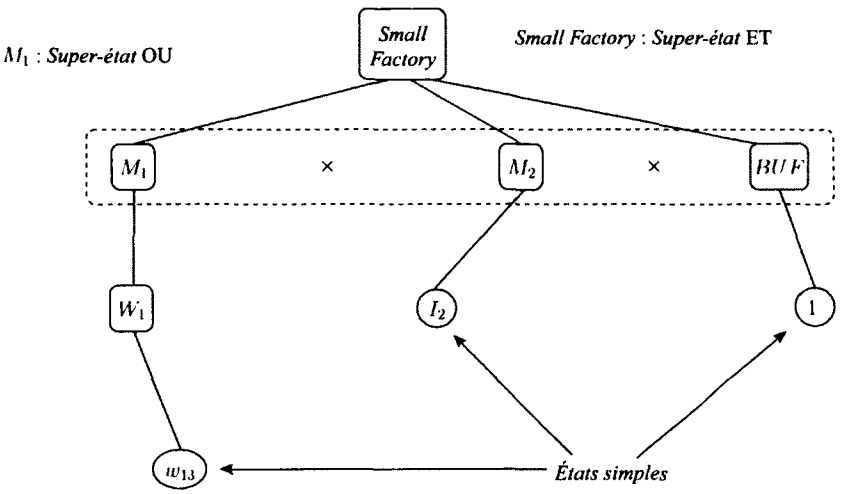


FIGURE 2.8 – Arborescence d'états élémentaire

L'ensemble des événements d'un holon est aussi partitionné en sous-ensembles d'événements de frontière Σ_B (*boundary events*) et d'événements internes Σ_I. La partition en événements contrôlables et incontrôlables (Σ_c, Σ_u) se superpose arbitrairement sur la partition en événements de frontière et événements internes (Σ_B, Σ_I) ; il n'y a pas de relation entre les deux. On a alors $\Sigma_B \cup \Sigma_I = \Sigma = \Sigma_c \cup \Sigma_u$ et $\Sigma_B \cap \Sigma_I = \emptyset = \Sigma_c \cap \Sigma_u$.

Ce partitionnement se reflète sur la fonction de transition partielle du holon δ, dont la figure 2.10 donne un sommaire. Les transitions d'entrée sont données par δ_{BI} (*boundary input transitions*) et les transitions de sortie par δ_{BO} (*boundary output transitions*). L'ensemble X_0 des états initiaux du holon ne peut pas être vide et contient tout état interne formant la cible d'une transition d'entrée dans la structure interne du holon. S'il n'y a aucune transition d'entrée, le contenu de X_0 est arbitraire, et on définit $\delta_{BI} : X_E \times \Sigma_B \rightarrow X_0$. Similairement, l'ensemble X_m des états finaux du holon ne peut pas être vide non plus, et contient tout état interne à l'origine d'une transition de sortie du holon. S'il n'y a aucune transition de sortie, le contenu de X_m est arbitraire et on définit $\delta_{BO} : X_m \times \Sigma_B \rightarrow X_E$. L'intersection de X_0 et X_m n'est pas nécessairement vide. Dans la figure 2.10, la structure interne d'un holon réfère donc aux objets X_I, Σ_I et δ_I et sa structure externe aux objets X_E, Σ_B et δ_B.

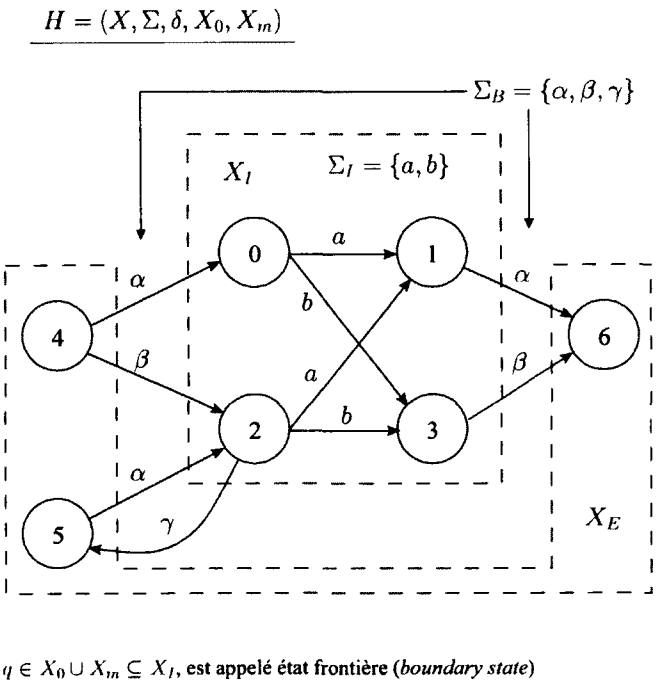


FIGURE 2.9 – Structure d'un *holon* ([39] figure 2.13)

$$H = (X, \Sigma, \delta, X_0, X_m)$$

$$\delta : X \times \Sigma \rightarrow X, \text{ où } \delta = \delta_I \cup \delta_B, \delta_I \cap \delta_B = \emptyset$$

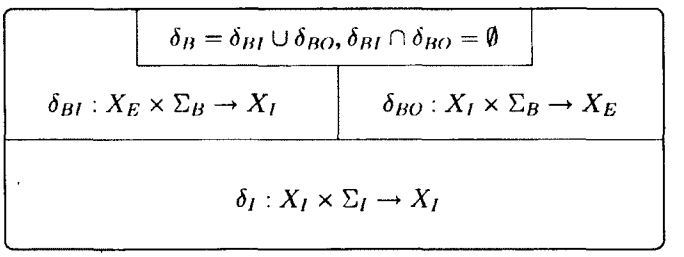


FIGURE 2.10 – Schéma de la relation de transition d'un *holon*

Pour assigner les éléments de dynamique locale (holons) aux éléments de la structure statique (les arborescences d'états), il faut assigner des holons à chaque super-état OU. Il est évident qu'un état simple n'a pas de dynamique associée, et que la dynamique d'un super-état ET est entièrement définie par la dynamique de ses composantes ; on ne considère ici que les ST bien formées. Or la structure arborescente des ST impose des contraintes entre les paires de holons assignés à des super-états OU et liés par une relation de filiation ($x < y$). Dans ce cas, il y a chevauchement entre la structure interne de x et la structure externe de y. On doit donc définir la relation d'ancêtre OU le plus rapproché (*Nearest OR ancestor*), puisque deux super-états OU tels que $x < y$ peuvent être séparés par un nombre arbitraire de super-états ET. Il faut donc imposer une règle garantissant la superposition consistante de leurs structures interne (de x) et externe (de y). Cette règle de superposition prend la forme de la propriété de *consistance à l'interface.*

Pour deux super-états OU tels que $x < y$ où x est l'ancêtre OU le plus rapproché de y, deux holons $H^x = (X^x, \Sigma^x, \delta^x, X_0^x, X_m^x)$ et $H^y = (X^y, \Sigma^y, \delta^y, X_0^y, X_m^y)$ se superposent à x et y respectivement si et seulement si ils sont consistants à l'interface, c'est-à-dire qu'ils sont consistants à l'interface quant à leurs états, leurs alphabets et leurs transitions d'entrée et de sortie (notion détaillée dans [39], pages 38–39). Pour de telles paires de

super-états OU et de holons (H^x, H^y) consistants à l'interface, on dit alors que H^y détaille (*expands*) H^x. Cette notion est illustrée intuitivement à la figure 2.11, où l'on voit que la structure de y détaille la structure sommaire de x, en ce sens que l'on peut substituer la version détaillée de y dans la structure sommaire de x sans qu'il n'y ait de contradiction à l'interface.

2.7.1.3 Structure en arborescence d'états

À partir de la notion d'arborescence d'états et du principe de superposition des holons, une définition de la notion de structure en arborescence d'états (STS) peut être élaborée. On définit $\mathbf{G} = (\mathbf{ST}, \mathcal{H}, \Sigma, \Delta, \mathbf{ST}_0, \mathbf{ST}_m)$, une structure en arborescence d'états comme une structure algébrique, où

- $\mathbf{ST} = (X, x_0, \mathcal{T}, \mathcal{E})$ est une arborescence d'états (celle dénotant tous les états du système) ;
- $\mathcal{H}$ est l'ensemble de holons superposés couvrant tous les super-états OU de $\mathbf{ST}$;
- Σ est l'ensemble des événements trouvés dans $\mathcal{H}$;
- $\Delta : ST(\mathbf{ST}) \times \Sigma \rightarrow ST(\mathbf{ST})$ est la fonction de progression dans $ST(\mathbf{ST})$;
- $\mathbf{ST}_0 \in ST(\mathbf{ST})$ est l'arborescence des états initiaux ;
- $\mathbf{ST}_m \subseteq ST(\mathbf{ST})$ est l'ensemble des arborescences marquantes de $\mathbf{G}$.

Une condition dite *condition de couplage local* doit être imposée pour permettre à deux super-états OU $x \mid y$ (c'est-à-dire en relation de parallélisme) de se synchroniser sur un même événement. À cette fin on définit la notion de ET-adjacence. Un noeud z est dit ET-adjacent à un noeud y, dénoté $z <_\times y$, si

1. $z < y$;
2. $\mathcal{T}(z) = and$;
3. et tout noeud situé entre z et y est un super-état ET.

Ainsi deux super-états OU $x \mid y$ ne peuvent se synchroniser lors d'une transition sur un événement σ (c'est-à-dire partager σ) que s'ils sont ET-adjacents à un même super-état ET. Cette situation correspond au mécanisme de synchronisation utilisé dans un modèle de composition synchrone (modèle plat).

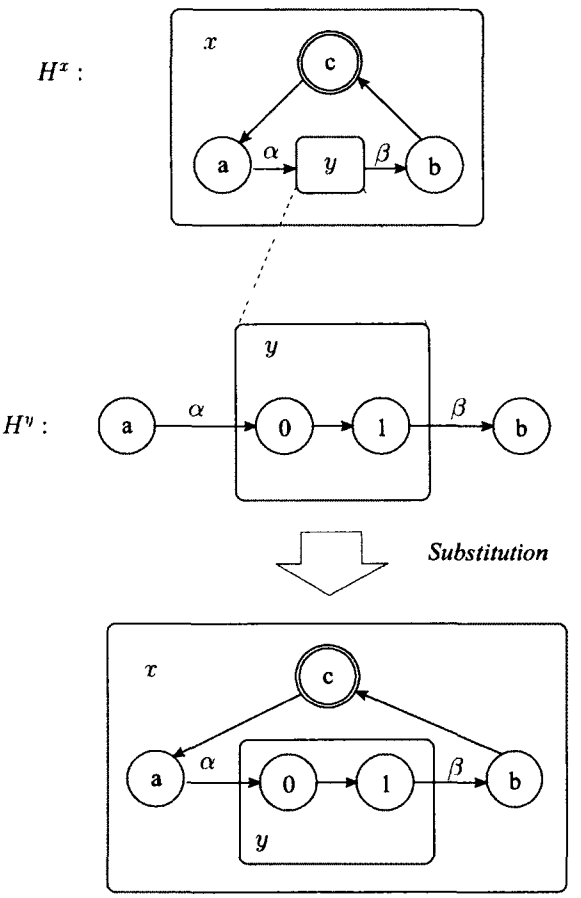


FIGURE 2.11 – Superposition de *holons* ([39] figure 2.17)

La dynamique globale du système peut être exprimée en termes de la dynamique locale de l'ensemble des holons $\mathcal{H}$. Ceci implique que Δ n'a pas à être explicitée pendant les calculs, ce qui représente une réduction significative de complexité pour les algorithmes. La fonction Δ est une fonction totale. Elle associe la plus grande arborescence cible possible dans $\mathbf{G}$, lors d'une transition sur $\sigma \in \Sigma$, à partir d'une arborescence source donnée.

En particulier, Δ associe tout élément de $\mathcal{B}(\mathbf{S}T)$ à un autre élément de $\mathcal{B}(\mathbf{S}T)$ (conséquence de la consistance aux interfaces), ou à l'arborescence vide $\emptyset$.

La fonction Δ est injective mais elle n'est pas surjective, elle n'a donc pas d'inverse. Cependant, il est possible de définir une fonction totale $\Gamma : ST(\mathbf{S}T) \times \Sigma \rightarrow ST(\mathbf{S}T)$ sur le même modèle que Δ, et telle que $\Delta(\Gamma(T, \sigma), \sigma) \leq T$. La fonction Γ est appelée fonction de progression rétrograde de $\mathbf{G}$. Elle permet de retrouver toutes les arborescences sources de $\mathbf{S}T$ dont la cible d'une transition sur $\sigma \in \Sigma$ est une arborescence contenue dans $T \leq \mathbf{S}T$. Cette fonction permet l'équivalent (chez les arborescences d'états) des calculs de co-accessibilité chez les AFD. Elle est la seule utilisée en synthèse, elle revêt donc une importance particulière.

Les détails de définition des fonctions Δ et Γ sont complexes et ne peuvent pas être exposés dans cette thèse sans l'allonger indûment. Le lecteur peut se référer à [39] pour les trouver. Ils sont accompagnés d'une description des conditions sous lesquelles le modèle de STS est équivalent à un modèle de composition synchrone (modèle dit RW dans l'ouvrage), c'est-à-dire des conditions sous lesquelles ces fonctions peuvent être considérées comme *justes* (de l'anglais *sound*). Le lecteur peut ainsi vérifier que ces conditions sont faciles à remplir et ne restreignent pas cette approche par rapport au modèle SCT original.

2.7.2 Synthèse utilisant des STS

La synthèse de contrôleurs utilisant les STS suit une approche par prédicats très proche de celle exposée à la section 2.5. Il y a donc d'importants recoupements.

On observe d'abord que toute sous-arborescence d'états $\mathbf{S}T_1 \in ST(\mathbf{S}T)$ peut être identifiée à un prédicat $P : \mathcal{B}(\mathbf{S}T) \rightarrow \{0, 1\}$ à travers un sous-ensemble d'arborescences d'états élémentaires

$$B_P = \{b \in \mathcal{B}(\mathbf{S}T) \mid P(b) = 1\} = \mathcal{B}(\mathbf{S}T_1) \subseteq \mathcal{B}(\mathbf{S}T),$$

où $\mathbf{S}T$ est l'arborescence d'états du système représenté par le STS $\mathbf{G}$. Ainsi $b \models P$ si et seulement si $b \in B_P$. Par extension, pour $\mathbf{S}T_1 \leq \mathbf{S}T$, $\mathbf{S}T_1 \models P$ si et seulement si $\mathcal{B}(\mathbf{S}T_1) \subseteq B_P$. On définit $Pred(\mathbf{S}T)$ l'ensemble des prédicats définissables sur $\mathbf{S}T$, en admettant par convention que pour tout $P \in Pred(\mathbf{S}T)$, $\emptyset \models P$, où $\emptyset$ est l'arborescence vide. Alors, la structure $(Pred(\mathbf{S}T), \preceq)$ forme un treillis complet.

Ce qui précède permet de reformuler la structure d'un STS comme une structure algébrique $\mathbf{G} = (\mathbf{S}T, \mathcal{H}, \Sigma, \Delta, P_0, P_m)$, où P_0 dénote $\mathbf{S}T_0$ l'arborescence des états initiaux, et P_m dénote

$$\bigcup_{T \in \mathbf{S}T_m} T.$$

En reformulant la notion de SFBC f sur $\mathcal{B}(\mathbf{S}T)$ et en l'appliquant sur le STS $\mathbf{G}$, on obtient le comportement du système en boucle fermée sous f, $\mathbf{G}^f = (\mathbf{S}T, \mathcal{H}, \Sigma, \Delta^f, P_0^f, P_m)$ où $P_0^f \preceq P_0$, avec Δ^f :

$$\Delta^f(b, \sigma) := \begin{cases} \Delta(b, \sigma) & \text{si } f_\sigma(b) = 1 \\ \emptyset & \text{autrement.} \end{cases}$$

La fonction Δ^f n'est définie que sur $\mathcal{B}(\mathbf{S}T)$.

La première nouveauté concerne la contrôlabilité. En effet on observe qu'il n'est pas nécessaire qu'un prédicat P soit accessible ($P \preceq R(\mathbf{G}, P)$) pour que l'on puisse obtenir une SFBC f qui implémente $R(\mathbf{G}, P)$, mais qu'il suffit qu'il soit invariant sous le flot des événements incontrôlables. En reformulant le foncteur $M_\sigma(\cdot)$, où $b \models M_\sigma(P)$ si et seulement si $\Delta(b, \sigma) \models P$, on formule la condition de contrôlabilité faible d'un prédicat (*weak controllability*). Un prédicat P est dit faiblement contrôlable si $P \preceq M_\sigma(P)$ pour tout $\sigma \in \Sigma$ (sans nécessairement être accessible).

Ainsi, pour un prédicat faiblement contrôlable P, lorsque $P \wedge P_0 \neq \mathit{false}$, il existe une SFBC f telle que $R(\mathbf{G}^f, true) = R(\mathbf{G}, P)$, notamment $f_\sigma = M_\sigma(P)$ quel que soit $\sigma \in \Sigma_c$. Ce résultat est très important car il permet, en synthèse, d'éviter l'évaluation de $R(\mathbf{G}, P)$, le prédicat d'accessibilité, ce qui représente une réduction de complexité considérable des calculs.

Pour la synthèse, on considère maintenant l'ensemble $\mathcal{CP}(P)$ des sous-prédicats de P faiblement contrôlables (plutôt que contrôlables). Cet ensemble est non vide et stable sous l'union, et sa borne supérieure est maintenant donnée par $\sup\mathcal{CP}(P) = \neg[\neg P]$ comme on

pouvait s'y attendre. Aussi, pour que $R(\mathbf{G}, \sup\mathcal{CP}(P))$ ne soit pas trivial, il faut bien sûr que $P_0 \preceq \neg[\neg P]$.

Les auteurs démontrent ensuite que pour un prédicat faiblement contrôlable et non bloquant P (sous condition que $P \wedge P_0 \neq false$), il existe une SFBC non bloquante f telle que $R(\mathbf{G}^f, true) = R(\mathbf{G}, P)$, définie par $f_\sigma = M_\sigma(P)$ pour tout $\sigma \in \Sigma_c$. Avec ce résultat et en observant qu'un prédicat co-accessible est nécessairement non bloquant, il est clair que si P est co-accessible et faiblement contrôlable (sous condition que $P \wedge P_0 \neq false$), alors il existe une SFBC non bloquante f pour $\mathbf{G}$, telle que $R(\mathbf{G}^f, true) = R(\mathbf{G}, P)$.

Du côté synthèse, on considère l'ensemble $\mathcal{C}^2\mathcal{P}(P)$ des sous-prédicats de P faiblement contrôlables et co-accessibles. On trouve que $\sup\mathcal{C}^2\mathcal{P}(P)$ existe et qu'il est exprimable comme point fixe de l'équation de récurrence

$$\Omega_P(K) := P \wedge CR(\mathbf{G}, \sup\mathcal{CP}(K)) = P \wedge CR(\mathbf{G}, \neg[\neg K]).$$

Si $P_0^f = P_0$ et $\sup\mathcal{C}^2\mathcal{P}(P) \neq false$, f s'exprime alors, pour tout $\sigma \in \Sigma_c$ et $b \in \mathcal{B}(\mathbf{ST})$, par

$$f_\sigma(b) := \begin{cases} 0 & \text{si } \Delta(b,\sigma) \models \neg \sup\mathcal{C}^2\mathcal{P}(P) \\ 1 & \text{autrement.} \end{cases}$$

2.7.3 Usage de BDD en synthèse avec des STS

La synthèse de contrôleurs par STS est particulièrement bien adaptée à l'utilisation de diagrammes de décision vu l'usage d'arborescences d'états pour représenter la structure statique du système. En effet, une arborescence d'états $\mathbf{ST}_1 \leq \mathbf{ST}$ quelconque de $\mathbf{G}$ est toujours identifiée à un prédicat $P : \mathcal{B}(\mathbf{ST}) \to \{0,1\}$ à travers son sous-ensemble d'arborescences d'états élémentaires $B_P = \mathcal{B}(\mathbf{ST}_1)$.

La théorie définit une projection $\Theta : ST(\mathbf{ST}) \to Pred(\mathbf{ST})$, pour une arborescence d'états quelconque $\mathbf{ST} = (X, x_0, \mathcal{T}, \mathcal{E})$, comme suit :

$$\Theta(\mathbf{ST}) := \begin{cases} \bigwedge\limits_{y \in \mathcal{E}(x_0)} \Theta(\mathbf{ST}^y) & \text{si } \mathcal{T}(x_0) = and \\ \bigvee\limits_{y \in \mathcal{E}(x_0)} ((v_{x_0} = y) \wedge \Theta(\mathbf{ST}^y)) & \text{si } \mathcal{T}(x_0) = or \\ 1 & \text{si } \mathcal{T}(x_0) = simple. \end{cases}$$

La définition suppose que chaque super-état OU (disons y) de $\mathbf{S}T$ a été associé à une variable d'état v_y dont le domaine est son expansion directe $\mathcal{E}(y)$. On peut noter que l'énumération en pré-ordre de ces variables sur $\mathbf{S}T$ constitue un ordonnancement adéquat pour la représentation de $\Theta(\mathbf{S}T_1)$ par un BDD pour $\mathbf{S}T_1 \leq \mathbf{S}T$. La définition de Θ elle-même est adéquate comme procédure d'encodage.

On peut vérifier en consultant les équations de synthèse que Θ résout la plupart des problèmes quant à l'usage de diagrammes de décision. Le seul problème qui reste est de déterminer comment les foncteurs $[\cdot]$ et $CR(\cdot)$ peuvent être implémentés à l'aide de BDD. On peut observer que ces deux foncteurs utilisent la fonction de progression rétrograde Γ pour calculer un point fixe dans l'espace $Pred(\mathbf{S}T)$. Par exemple, l'algorithme 4.1 dans [39] donne la solution suivante pour le calcul de $[R]$.

1. $K_0 := R$; $i := 0$;
2. $K_{i+1} := K_i \vee \left(\bigvee_{\sigma \in \Sigma_u} \Gamma(K_i, \sigma) \right)$;
3. si $K_{i+1} \equiv K_i$ alors $[R] = K_i$, sinon reprendre à l'étape (2) avec $i := i + 1$.

Le calcul de $CR(\cdot)$ utilise un schéma similaire où seule l'évaluation de Γ reste problématique. Il faut bien sûr exprimer $\hat{\Gamma} : Pred(\mathbf{S}T) \times \Sigma \rightarrow Pred(\mathbf{S}T)$, une version prédicative de $\Gamma : ST(\mathbf{S}T) \times \Sigma \rightarrow ST(\mathbf{S}T)$. La solution donnée est la suivante :

$$\hat{\Gamma}(P, \sigma) := (\exists \mathbf{v}_{\sigma,\mathbf{T}}(P \wedge N_\sigma))[\mathbf{v}'_{\sigma,\mathbf{S}} \rightarrow \mathbf{v}_{\sigma,\mathbf{S}}].$$

Dans cette équation, N_σ doit encoder, sous la forme d'un prédicat, la relation de transition, c'est-à-dire l'ensemble des transitions possibles sur $\sigma \in \Sigma$ dans $\mathbf{G}$. On a déjà vu que l'encodage de la relation de transition est un problème très épineux en vérification symbolique [26].

On observe que dans une structure en arborescence d'états, σ n'apparaîtra que localement dans les holons H^x qui partagent cet événement. Chaque transition peut être décrite par son effet local dans le holon H^x. Sommairement, pour un super-état OU x dans une arborescence d'états quelconque $\mathbf{S}T$ où au moins une transition sur $\sigma \in \Sigma$ peut se produire, le sous-arbre $\mathbf{S}T_1^x$ enraciné au noeud x subit une transformation en un autre sous-arbre $\mathbf{S}T_2^x$ sous l'effet de la transition. Ce changement est limité au sous-arbre $\mathbf{S}T^x$ de $\mathbf{S}T$. En posant

$\Gamma(\mathrm{S}T', \sigma) = \mathrm{S}T$, on a que $\Theta(\mathrm{S}T')$ est le prédicat dénotant les cibles des transitions sur σ dans $\mathrm{S}T$. Puisque $\mathrm{S}T'$ contient nécessairement $\mathrm{S}T_2^x$ dans sa structure, $\Theta(\mathrm{S}T')$ contient aussi $\Theta(\mathrm{S}T_2^x)$ dans sa structure. En substituant $\Theta(\mathrm{S}T_2^x)$ dans $\Theta(\mathrm{S}T')$ par $\Theta(\mathrm{S}T_1^x)$, on obtient $\Theta(\mathrm{S}T_1)$ où $\mathrm{S}T_1 \leq \mathrm{S}T$ contient toutes les sources de transitions sur σ avec cible dans $\mathrm{S}T'$.

Mais bien sûr pour exprimer «$v_x = 1$ avant l'occurrence de σ» **et** «$v_x = 2$ après l'occurrence de σ» dans un seul et même prédicat, on ne peut utiliser la même variable v_x. On a donc recours à deux variables liées entre elles : v'_x (*variable primée*) et v_x (*variable ordinaire*) dénotant respectivement l'état avant et après l'occurrence de σ. Ainsi, pour exprimer cette transition (appelons la **t**), où $T = \mathrm{S}T_2^x$ désigne la cible et $S = \mathrm{S}T_1^x$ désigne la source, en termes d'un prédicat sur $ST(\mathrm{S}T)$, on peut utiliser la conjonction suivante $N_t := \Theta(S)[\mathbf{v}_{\mathbf{t},\mathrm{S}} \rightarrow \mathbf{v}'_{\mathbf{t},\mathrm{S}}] \wedge \Theta(T)$. Dans cette conjonction $[\mathbf{v}_{\mathbf{t}.\mathrm{S}} \rightarrow \mathbf{v}'_{\mathbf{t},\mathrm{S}}]$ dénote simplement que l'ensemble des variables ordinaires relatives à **t** dans S sont renommées en leurs correspondantes primées. La forme de N_t n'est pas tout à fait exacte car on doit aussi tenir compte du reste de l'arborescence au-dessus de $\mathrm{S}T^x$ jusqu'à la racine. Notons simplement qu'une version exacte n'est pas difficile à obtenir en complétant le chemin qui mène de x_0 à x dans $\mathrm{S}T$. Par la suite, obtenir N_σ consiste à combiner tous les prédicats de transitions N_t sur σ qui peuvent se produire dans G en utilisant des disjonctions (plusieurs transitions dans un même holon) et des conjonctions (transitions synchronisées entre deux ou plusieurs holons dans G) entre prédicats, au besoin.

À noter que N_σ est un prédicat défini sur les deux ensembles de variables (primées et ordinaires). Le résultat de la conjonction $P \wedge N_\sigma$ est donc un prédicat exprimé sur l'ensemble des variables primées et ordinaires. Il exprime l'effet d'une progression rétrograde sur σ dans G. Or la quantification sur les BDD a pour effet d'éliminer la variable quantifiée. Pour un ensemble de variables $\mathbf{v} = \{\mathbf{v}_1, ..., \mathbf{v}_n\}$, on a $(\exists \mathbf{v})\, P = (\exists \mathbf{v}_1 (\exists \mathbf{v}_2 ... (\exists \mathbf{v}_n P) ...))$. La quantification existentielle élimine donc ici toutes les variables ordinaires du prédicat P correspondant aux cibles d'une transition sur σ. On retrouve donc comme résultat un prédicat exprimant les sources de ces transitions mais exprimées en termes de variables primées. Ainsi, la dernière opération dans l'expression de $\hat{\Gamma}$, $[\mathbf{v}'_{\sigma,\mathrm{S}} \rightarrow \mathbf{v}_{\sigma,\mathrm{S}}]$, consiste à renommer les variables primées en variables ordinaires correspondantes.

En conclusion, on peut noter qu'il est possible de donner un équivalent à ce schéma

de calcul pour un modèle de composition synchrone (voir [63]). Mais comme en fait foi la littérature, le calcul de la relation de progression fait souvent appel à des algorithmes complexes. Aussi, puisque ces techniques ne font usage que d'AFD, l'évaluation exhaustive de la fonction de transition globale de l'AFD semble inévitable. La relation de transition N_σ doit alors tenir compte de l'état de toutes les variables lors d'une transition, ce qui induit beaucoup de redondance, c'est-à-dire d'expressions du style $\mathbf{v}'_\mathbf{x} = \mathbf{v}_\mathbf{x}$. Ceci allonge significativement les calculs et peut induire une grande consommation en espace.

Un modèle de structure en arborescence d'états peut représenter directement un modèle de composition synchrone (modèle plat) comme dans l'exemple de *Small Factory* de la figure 2.6. sans encourir les pénalités bien connues en vérification. Aussi, [39] donne tous les algorithmes de base, bien que sous forme abstraite, et sans jamais donner de solution concrète pour l'implémentation par BDD. Plusieurs optimisations aux algorithmes sont aussi suggérées pour tenir compte de la taille des modèles (prédicats) intermédiaires durant les calculs. Ces suggestions sont très proches de techniques utilisées en *model-checking* tel le concept de saturation [6].

Finalement, le résultat de la synthèse avec les STS est un ensemble de prédicats f_σ que l'on peut utiliser pour implémenter une SFBC, avec seul désavantage qu'ils sont exprimés sous forme de BDD. Ceci suppose un ordinateur d'assez grande puissance pour le contrôle. Sans compter que le problème de suivre la configuration courante du système (c'est-à-dire l'arborescence $b \in \mathcal{B}(ST)$ dans lequel le système se trouve à tout moment) n'est pas très évidente. Cependant, sur une note plus positive, l'usage d'une SFBC dans cette approche n'est pas restrictif puisque la description du système incorpore les éléments de mémoire, par exemple l'usage d'un tampon (BUF) dans le modèle *Small Factory*.

2.8 Approche par agrégation d'états

Toutes les variantes hiérarchiques de SCT supposent nécessairement une forme d'agrégation d'états.

- Dans l'approche par STS (section 2.7), l'agrégation d'états est explicite sous forme de *super-états*.
- Dans l'approche HISC (section 2.4), l'agrégation d'états est implicite dans la struc-

ture d'interfaces en paires commande-réponse.

- Dans l'approche proposée pour le contrôle hiérarchique de la section 2.3, l'agrégation d'états est implicite dans la structure du système de haut niveau $\mathbf{G}_{hl}$.

Cette section décrit deux approches d'abstraction du modèle (AFD) d'un système. Elles tentent d'identifier ou de calculer un AFD *quotient* dont les états sont identifiés aux éléments d'une partition de l'espace des états de l'AFD du système original (une relation d'équivalence entre ces états). Pour être utilisés dans les calculs de synthèse, les quotients considérés doivent, en plus d'être déterministes, présenter certaines propriétés. Hubbard et Caines [29] définissent une structure de quotient d'AFD particulière, appelé *automate partition*, devant posséder une propriété appelée la *consistance dynamique*. L'approche de Wong et Wonham [60] ainsi que celle de Feng et Wonham [15] s'inscrivent dans le cadre du modèle du contrôle hiérarchique (section 2.3.2). Cette approche tente de *raffiner* le noyau de la projection $\theta : L \rightarrow M$ d'un modèle donné a priori, sous forme d'une machine de Mealy, de la paire (L, θ) de façon à doter la projection de la propriété d'observateur (voir la section 2.3.3).

L'intuition à la base de ces approches est de tenter d'identifier des chaînes de partition de granularité croissante de l'espace des états d'un système qui permettent d'exprimer les requis de contrôle à différents niveaux en plus de permettre la synthèse. Une telle technique procure une clarification significative dans l'expression des requis d'un système en permettant l'*encapsulation* par *niveau de portée*, c'est-à-dire en exprimant d'abord les requis de portée locale, puis en progressant par encapsulation successive vers les requis de portée de plus en plus globale. Cette technique vise aussi bien sûr à tenter de réduire la complexité des calculs de synthèse.

De plus, pour les systèmes où la loi de contrôle peut s'exprimer sous forme d'une SFBC (voir la section 2.5), l'usage de partitions (des sous-ensemble d'états) comporte un avantage évident. Les prédicats d'une SFBC décrivent des familles d'états du système sous contrôle, ce sont donc des *indicatrices* ou *fonctions caractéristiques* dénotant des sous-ensembles d'états. En supposant un résultat de synthèse sur un modèle abstrait (lire *agrégé*) du système on obtient une SFBC *abstraite*, c'est-à-dire dont les termes élémentaires dénotent des états du *modèle abstrait*. Or dans une structure d'AFD *quotient* ces états du modèle abstrait forment une *partition* de l'ensemble des états du modèle concret, et peuvent donc eux-

mêmes être exprimés sous la forme d'*indicatrices* de sous-ensembles du *modèle concret*. Ce qui procure une base simple pour le calcul d'une loi de contrôle sur le modèle concret à partir d'un résultat de synthèse sur le modèle abstrait.

Ces approches utilisent la notion de *projections causales* (voir la section 2.3.2.1) au même titre que le contrôle hiérarchique (section 2.3), avec les mêmes avantages. Le système de bas niveau (l'agent) *projette* son comportement sur un autre alphabet (celui de l'abstraction), effaçant des transitions et générant des événements sommaires ayant une signification plus appropriée dans l'abstraction, dans le but d'obtenir une description simplifiée sans sacrifier la structure de contrôle et la possibilité de synthèse. Cependant, à la différence du contrôle hiérarchique tel qu'exposé dans Wong [55] et Wonham [62], il n'est pas nécessaire d'enrichir la structure de transition de l'agent (qui est alors transformé en *machine de Moore*) pour obtenir la consistance du contrôle (voir la section 2.3.2.2). Ces approches se limitent à l'usage des projections que l'on peut obtenir à partir de la structure de transition de l'AFD du système original (l'agent) que l'on transforme en machine de Mealy en lui adjoignant un alphabet de sortie et une projection appropriée à chaque transition.

2.8.1 Calcul d'observateurs d'un SED

Cette technique repose sur la notion de projection causale que l'on peut implémenter par marquage des *transitions* d'un AFD résultant en une machine de Mealy comme exposé dans la section 2.3.3. On considère que l'ensemble des objets et notions définis dans cette section sont connus. Ce qui suit n'est forcément qu'une illustration informelle tirée, pour l'essentiel, de Wong et Wonham [60] que le lecteur est invité à consulter.

2.8.1.1 Autobisimulation et quasi-congruence

Comme mentionné dans [58] la propriété d'observateur est une relation d'*équivalence observationnelle*. Et d'après Arnold [3], l'*équivalence observationnelle* peut se définir comme la *bisimulation modulo* ϕ_τ ; un *critère d'observation*. Dans le cas de l'équivalence observationnelle, l'alphabet (disons T) des deux systèmes est identique et le critère d'ob-

servation utilisé est

$$\begin{aligned} \phi_{\tau_0}^{-1}(\tau_0) &= \tau_0^* \\ \phi_{\tau_0}^{-1}(\tau) &= \tau_0^* \tau \tau_0^* \end{aligned}$$

où $\tau \in T$ est observable et $\tau_0 \notin T$ n'est pas observable. Toujours d'après Arnold, l'*autobisimulation* est une classe spéciale de bisimulation d'un système avec lui-même. La plus grande autobisimulation induite par une relation d'équivalence R et incluse dans R existe (ici R est définie sur l'ensemble des états d'un système) et peut être calculée dans le cas où R est finie. Dans ce cas, le calcul se ramène à un problème de raffinement de partition sur l'ensemble des états du système décrit en détail (avec algorithmes) dans Paige et Tarjan [41] et Fernandez [16].

La propriété d'observateur est décrite dans Wong et Wonham [60] comme une quasi-congruence sur le système dynamique $\widetilde{\mathbf{T}} = (X, \{\Xi_\tau \mid \tau \in T\} \cup \{\Xi_m\}, x_0, X_m)$:

$$\pi^* = sup \left\{ \pi \in \mathcal{E}(X) \mid \pi \leq (\ker \theta) \wedge (\pi \circ pre) \wedge \left(\bigwedge_{\tau \in T} (\pi \circ \Xi_\tau) \right) \wedge (\pi \circ \Xi_m) \right\} \quad (2.8)$$

où $\ker \theta$ et pre correspondent respectivement à la relation d'équivalence et à l'opérateur de fermeture préfixée tels qu'induits sur X à travers la bijection $f : \overline{L_m} \rightarrow X$ (voir la section 2.3.3.4). Posons

$$\pi := \left((\ker \theta) \wedge (\pi \circ pre) \wedge \left(\bigwedge_{\tau \in T} (\pi \circ \Xi_\tau) \right) \wedge (\pi \circ \Xi_m) \right).$$

Alors $\pi \in \mathcal{E}(X)$ est une relation d'équivalence sur X et la plus grande autobisimulation (du système $\widetilde{\mathbf{T}}$) induite par π et incluse dans π existe. Alors moyennant que l'on puisse se ramener à un cas fini, il est possible de la calculer par une variation de l'algorithme de Fernandez.

On suppose que l'on dispose de

$$\mathbf{G} = (Q, \Sigma, T \cup \{\tau_0\}, \delta, \lambda, q_0, Q_m)$$

un générateur de Mealy fini modélisant (L, θ) duquel on obtient $\widehat{\mathbf{G}}$ et le système dynamique $\widetilde{\mathbf{G}}$, correspondant au système dynamique $\widetilde{\mathbf{T}}$. En suivant l'argumentation de [60] on définit

$$h : X \to Q : x \mapsto \delta(q_0, f^{-1}(x))$$

et on peut vérifier que $q_0 = h(x_0)$, $Q_m = h(X_m)$ et

$$\begin{aligned} (\forall \tau \in T)\, h \circ \Xi_\tau &= \Delta_\tau \circ h \\ h \circ \Xi_m &= \Delta_m \circ h \end{aligned}$$

c'est-à-dire que h est un homomorphisme de $\widetilde{\mathbf{T}}$ à $\widetilde{\mathbf{G}}$. En considérant $\pi_{Mealy} \in \mathcal{E}(L)$ la congruence à droite correspondant à $\mathbf{G}$ et définie par

$$s \equiv s'(mod\ \pi_{Mealy}) \iff \delta(q_0, s) = \delta(q_0, s')$$

pour $s, s' \in L = \overline{L_m}$, que l'on induit ensuite sur X à l'aide de la bijection f, on peut vérifier que $\ker h = \pi_{Mealy}$, ce qui nous ramène au cas fini. Ainsi, lorsqu'on dispose d'un modèle fini pour (L, θ) (notamment la machine de Mealy $\mathbf{G}$ de laquelle on obtient $\widehat{\mathbf{G}}$ et $\widetilde{\mathbf{G}}$), la relation d'équivalence correspondant à la plus grande autobisimulation induite par $\rho \in \mathcal{E}(Q)$ et incluse dans ρ (pour le système dynamique $\widetilde{\mathbf{G}}$) correspond à la quasi-congruence

$$\rho_{\mathbf{G}} := sup \left\{ \rho \in \mathcal{E}(Q) \mid \rho \leq \left(\bigwedge_{\tau \in T} (\rho \circ \Delta_\tau) \right) \wedge (\rho \circ \Delta_m) \right\} \tag{2.9}$$

sur $\widetilde{\mathbf{G}}$. Cette quasi-congruence peut être calculée en utilisant une variation approprié de l'algorithme de raffinement de partition de Fernandez. Puisqu'il s'agit d'une relation d'équivalence sur Q, l'algorithme produit une partition de Q comme résultat, et l'on réfère à $\rho_{\mathbf{G}}$ en tant que tel. L'algorithme de Fernandez doit être adapté, entre autres, pour tenir compte de l'information de marquage. On trouve le développement formel détaillé de cette adaptation dans Wong et Wonham [60] et des détails d'implémentation dans Paige et Tarjan [41] et Fernandez [16].

2.8.1.2 Quotients d'automates finis et observateurs

Dans ce qui précède, $\rho_{\mathbf{G}}$ correspond à la congruence de Nérode sur $\theta(L)$ et il se peut que $\rho_{\mathbf{G}} < \ker\theta$. Aussi le calcul de $\rho_{\mathbf{G}}$ ne constitue-t-il qu'une itération de l'algorithme de Wong. L'algorithme de Wong consiste à raffiner une projection causale $\theta : L \to M$ arbitraire jusqu'à ce que $\ker\theta = \pi^*$. À cette fin on doit disposer d'un test pour déterminer quand $\ker\theta = \pi^*$ et ce test est basé sur la notion de quotient d'automates.

Soit $\mathbf{H} = (Q, \Sigma, \delta, q_0, Q_m)$ un automate non déterministe. Pour $\Sigma' \subseteq \Sigma$ et $Q' \subseteq Q$ on définit

$$\delta_{\Sigma'}(Q') := \bigcup \{\delta(q,\sigma) \mid q \in Q', \sigma \in \Sigma'\} \tag{2.10}$$

et par abus de notation on utilise δ_σ lorsque $\Sigma' = \{\sigma\}$. Pour $\rho \in \mathcal{E}(Q)$ et $Y := Q/\rho$, l'ensemble quotient, on définit $g : Q \to Y : q \mapsto [q]_\rho$ la projection canonique. Aussi on définit $y_0 := g(q_0)$, l'état initial, et $Y_m := g(Q_m)$ l'ensemble des états marqués du quotient. Posons un symbol spécial $\tau_0 \notin \Sigma$ et $\Sigma_o \subseteq \Sigma$ un ensemble d'événements *observables* par opposition à $\Sigma - \Sigma_o$ l'ensemble des événements considérés inobservables pour les besoins de l'argumentation. Finalement on définit la relation de transition *induite* suivante :

$$\eta : Y \times (\Sigma_o \cup \{\tau_0\}) \to \mathcal{P}(Y) :$$
$$(y,\sigma) \mapsto \begin{cases} g(\delta_\sigma(g^{-1}(y))), & \text{si } \sigma \in \Sigma_o\,; \\ g(\delta_{(\Sigma-\Sigma_o)}(g^{-1}(y))) - \{y\}. & \text{autrement } (\sigma \text{ est inobservable}) \end{cases} \tag{2.11}$$

où τ_0 est l'étiquette d'une transition inobservable. On définit le quotient $\mathbf{H}/(\Sigma_o, \rho)$ par

$$\overline{\mathbf{H}} = \mathbf{H}/(\Sigma_o, \rho) := (Y, \Sigma_o \cup \{\tau_0\}, \eta, y_0, Y_m) \tag{2.12}$$

et $\overline{\mathbf{H}}$ n'est généralement pas déterministe. On peut vérifier que si $\mathbf{H}$ est un automate soigné (section 1.1.3) il en va de même pour $\overline{\mathbf{H}}$.

On définit

$$g_t : \mathcal{T}_\delta \to \mathcal{T}_\eta :$$
$$(q,\sigma,q') \mapsto \begin{cases} (g(q),\sigma,g(q')), & \text{si } \sigma \in \Sigma_o\,; \\ (g(q),\tau_0,g(q')), & \text{si } \sigma \notin \Sigma_o \text{ et } g(q) \neq g(q')\,; \\ \textit{indéfinie}, & \text{autrement} \end{cases} \tag{2.13}$$

la fonction de correspondance des transitions entre $\mathbf{H}$ et $\mathbf{H}/(\Sigma_o,\rho)$. Donc g_t produit l'ensemble des transitions de $\overline{\mathbf{H}}$ à partir de l'ensemble des transitions de $\mathbf{H}$ de la façon suivante.

1. Les transitions de $\mathbf{H}$ dont l'événement est inobservable et dont les *classes d'équivalence dans* Y des états source et cible *sont les mêmes*, ne sont pas retenues.
2. Parmi les transitions de $\mathbf{H}$ projetées, les états source et cible sont remplacés par leur classe d'équivalence dans Y et le *même événement est conservé* s'il est observable ($\sigma \in \Sigma_o$) ou remplacé par τ_0 *s'il n'est pas observable* ($\sigma \in \Sigma - \Sigma_o$).

Aussi, $D_{g_t}(\mathcal{T}_\delta)$ dénote l'ensemble des transitions de $\mathbf{H}$ sur lesquelles g_t est définie, c'est-à-dire

$$D_{g_t}(\mathcal{T}_\delta) := \{\zeta \in \mathcal{T}_\delta \mid g_t(\zeta) \textit{ est définie}\} \tag{2.14}$$

de sorte que $(q,\sigma,q') \in \mathcal{T}_\delta - D_{g_t}(\mathcal{T}_\delta)$ si et seulement si $\sigma \in \Sigma_o$ et $g(q) = g(q')$. En clair, les transitions sur un événement inobservable (au sein de $\mathbf{H}$) qui sont internes à un même état du quotient, ne sont pas définies dans le quotient.

Soit maintenant $\widehat{\mathbf{G}} = (Q, \Sigma \cup T, \widehat{\delta}, q_0, Q_m)$ un automate fini obtenu par réétiquetage de $\mathbf{G}$, une machine de Mealy finie et déterministe représentant (L,θ) selon les équations 2.4 et 2.3. Soit $\rho_{\mathbf{G}}$ telle que définie par l'équation 2.9 et

$$\overline{\mathbf{G}} := \widehat{\mathbf{G}}/(T,\rho_{\mathbf{G}}) = (Y, T \cup \{\tau_0\}, \eta, y_0, Y_m)$$

où $\tau_0 \notin \Sigma \cup T$. Dans ce quotient on appelle une τ_0-transition une transition correspondant à $\eta(y,\tau_0)$ définie pour un certain $y \in Y$. Le résultat suivant caractérise la propriété d'observateur et procure un critère d'arrêt pour l'algorithme de Wong.

Théorème 4.1. *[60]*
La projection $\theta : L \to M$ est un L_m-observateur si et seulement si $\overline{\mathbf{G}}$ est déterministe et

ne contient aucune τ_0-transition.

L'automate $\widehat{\mathbf{G}}$ initial défini d'après l'équation 2.4 sert de point de départ à l'algorithme de Wong et suit le plan suivant.

1. Calculer ρ_{i+1}, la quasi-congruence la plus grossière correspondant à $\widetilde{\mathbf{G}}_i$.
2. Calculer $\overline{\mathbf{G}}_i = \widehat{\mathbf{G}}_i/(T_i, \rho_{i+1})$.
3. Si $\overline{\mathbf{G}}_i$ est déterministe et libre de toute τ_0-transition, alors $\overline{\mathbf{G}} := \overline{\mathbf{G}}_i$ et on arrête ; sinon on continue à l'étape suivante.
4. Réétiqueter $\widehat{\mathbf{G}}_i$ de sorte à rendre $\overline{\mathbf{G}}_i$ déterministe et libre de τ_0-transitions en ajoutant de nouveau événements à T_i si nécessaire, donnant un nouvel alphabet T_{i+1} et un nouvel automate $\widehat{\mathbf{G}}_{i+1}$, incrémenter i et retourner à l'étape (1).

En sortie on retrouve un quotient déterministe $\overline{\mathbf{G}}$. L'étape de réétiquetage peut détruire la relation de quasi-congruence, c'est pourquoi on doit la recalculer après un réétiquetage. La logique de réétiquetage peut éventuellement être adaptée moyennant certaine restrictions pour que l'algorithme converge. D'abondants détails peuvent être trouvés dans Wong et Wonham [60].

Le quotient obtenu est un AFD *minimal* reconnaissant le langage $\theta(L_m)$ et le calcul s'effectue en temps polynomial.

2.8.1.3 Lien avec le contrôle hiérarchique

Moyennant qu'il soit possible d'induire une technologie de contrôle adéquate dans le quotient obtenu par l'algorithme de Wong, d'assurer une forme de consistance du contrôle (section 2.3.2.2) et de réunir les conditions pour le contrôle non bloquant (section 2.3.4), les avantages de cette technique pour la conception de composantes réutilisables semblent évidents, en particulier pour un contrôle du type SFBC (section 2.5 et section 2.7).

2.8.2 Notion d'automate partition

Pour un AFD $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ et une partition π arbitraire sur son espace d'états Q, on peut former un automate-partition (π-AFD) $\mathbf{G}^\pi = (\pi, \Sigma^\pi, \delta^\pi, Q_0^\pi, Q_m^\pi)$, où

- $\pi = \{Q_1, \ldots, Q_n\}$, $n > 1$, est une partition de Q représentant les états de $\mathbf{G}^\pi$;

- $\Sigma^\pi = \{\tau_i^j \mid (\exists q, \sigma \mid q \in Q_i, \sigma \in \Sigma : \delta(q,\sigma)! \wedge \delta(q,\sigma) \in Q_j \neq Q_i)\}$ dénote un ensemble d'événements de transition entre les éléments de π ;
- $\delta^\pi : \pi \times \Sigma^\pi \rightarrow \pi$ est une fonction de transition partielle définie par

$$\delta^\pi(Q_i, \tau_i^j) := \begin{cases} Q_j & \text{si } (\exists q, \sigma \mid q \in Q_i, \sigma \in \Sigma : \delta(q,\sigma)! \wedge \delta(q,\sigma) \in Q_j \neq Q_i) \\ \textit{indéfinie} & \text{autrement.} \end{cases}$$

Il n'est pas nécessaire de se soucier de Q_0^π et Q_m^π. Il suffit de savoir qu'ils existent et sont définis dans [29]. L'automate $\mathbf{G}^\pi$ est un AFD dont chaque état correspond à une famille d'états de G, et qui *signale* toute transition entre deux éléments distincts de la partition π.

On peut ainsi définir une projection $\Theta^\pi : Q \times \Sigma \rightarrow \Sigma^\pi \cup \{\varepsilon\}$ telle que, pour tout $q \in Q_i$ et $\sigma \in \Sigma$,

$$\Theta^\pi(q,\sigma) := \begin{cases} \tau_i^j & \text{si } \delta(q,\sigma)! \text{ et } \delta(q,\sigma) \in Q_j \neq Q_i \\ \varepsilon & \text{autrement.} \end{cases}$$

Cette projection (illustrée en figure 2.12) peut être *implémentée* en se servant de G pour former le générateur de Mealy $\mathbf{G}_{\mathbf{\Theta}^\pi} = (Q, \Sigma, \Sigma^\pi, \delta, \Theta^\pi, Q_0, Q_m)$. Il est évident que $\mathbf{G}_{\mathbf{\Theta}^\pi}$ génère $\Theta^\pi(L(\mathbf{G}))$ en sortie et que $\Theta^\pi(L(\mathbf{G})) \subseteq L(\mathbf{G}^\pi)$. Mais puisque π est arbitraire sur Q, on aura en général une inclusion stricte. En d'autres termes, même s'il était possible de doter $\mathbf{G}^\pi$ d'une structure de contrôle adéquate, $\mathbf{G}^\pi$ ne serait pas *hiérarchiquement consistant* en général, et il serait à toute fin pratique impossible de l'utiliser pour la synthèse de contrôleurs.

Les travaux de Caines et Hubbard exposés dans [29] consistent à identifier les conditions sous lesquelles un π-AFD du type de $\mathbf{G}^\pi$ peut être doté d'une structure de contrôle et garantir la propriété de consistance hiérarchique de sorte qu'il puisse éventuellement être utilisé en synthèse de contrôleurs. Ils utilisent à cette fin un cas particulier de *superstructure cellulaire* telle que définie par Schwartz [48] dont la dynamique est représentée par un π-AFD du type de $\mathbf{G}^\pi$.

Dans ce qui précède, on peut considérer que la partition π induit (dans G) un ensemble de sous-automates $\mathcal{G}_i = (Q_i \cup Q_{i,ex}, \Sigma, \delta_i, Q_{i,0}, Q_{i,ex})$ appelés *cellules*, où

- $Q_{i,ex} = \{q \in Q \setminus Q_i \mid \delta(q',\sigma) = q, q' \in Q_i, \sigma \in \Sigma\}$ est un ensemble d'états cibles externes à la cellule ;
- δ_i, la restriction de δ à $Q_i \times \Sigma$, est la fonction partielle de transition au sein de la

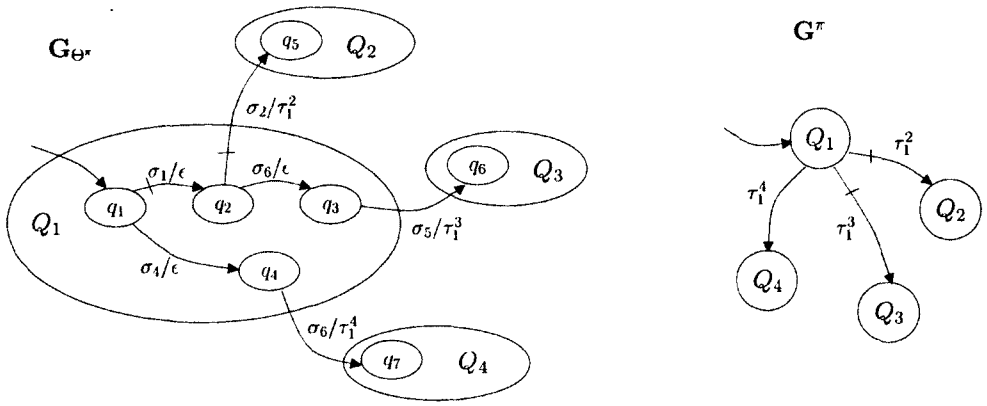


FIGURE 2.12 – Générateur de Mealy pour G^π

cellule ;

- $Q_{i,0} = \{q \in Q_i \mid \delta(q', \sigma) = q, q' \in Q_i \setminus Q, \sigma \in \Sigma\}$ est l'ensemble des états d'entrée dans la cellule.

Encore une fois on peut négliger la présence d'états initial et finaux ; les définitions détaillées sont données dans [29]. À chaque élément de la partition $Q_i \in \pi$, correspond une et une seule cellule $\mathcal{G}_i$.

À l'aide de cette notion de cellule, on peut définir une relation de portée entre deux cellules et, par extension, entre deux éléments de π. De façon informelle, la cellule $\mathcal{G}_j$ est dite être en portée de la cellule $\mathcal{G}_i$ pour $i \neq j$, et noté $\langle \mathcal{G}_i, \mathcal{G}_j \rangle$ si $Q_{i,ex} \cap Q_{j,0} \neq \emptyset$, ou encore si $CR_{\mathcal{G}_i}(Q_{j,0}) \neq \emptyset$. Par extension on dit alors aussi que Q_j est en portée de Q_i, noté $\langle Q_i, Q_j \rangle$.

D'une façon similaire on peut définir deux autres relations de portée. Notamment la portée incontrôlable,

$$\langle Q_i, Q_j \rangle_u \Leftrightarrow Q_{i,0} \subseteq CR^{\Sigma_u}_{\mathcal{G}_i}(Q_{j,0}),$$

et la portée contrôlable,

$$\langle Q_i, Q_j \rangle_d \Leftrightarrow (Q_{i,0} \subseteq CR_{\mathcal{G}_i}(Q_{j,0})) \wedge (Q_{i,0} \cap CR^{\Sigma_u}_{\mathcal{G}_i}(Q_{j,0}) = \emptyset).$$

Dans un π-AFD, $\langle Q_i, Q_j \rangle_u$ dénote que peu importe l'état d'entrée dans l'état Q_i ($q \in Q_{i,0}$), on peut toujours atteindre un état de $Q_{j\,0}$ (c'est-à-dire un état adjacent) par une trace formée uniquement d'événements incontrôlables de $\mathbf{G}$. Tandis que pour pouvoir inhiber une transition entre Q_i et Q_j (c'est-à-dire pour que la portée $\langle Q_i, Q_j \rangle_d$ soit réalisée), il faut que tous les états de $Q_{i\,0}$ soient liés à $Q_{j,0}$ et que ces traces soient toutes composées d'au moins un événement contrôlable de $\mathbf{G}$. Naturellement les portées $\langle Q_i, Q_j \rangle_u$ et $\langle Q_i, Q_j \rangle_d$ ne peuvent pas être réalisées simultanément.

On dit qu'un π-AFD $\mathbf{G}^\pi$ est *consistant quant aux traces* (de l'anglais *Trace-DC*) si, pour toute paire d'états $Q_i, Q_j \in \pi$, $i \neq j$,

$$\langle Q_i, Q_j \rangle \Rightarrow \langle Q_i, Q_j \rangle_u \vee \langle Q_i, Q_j \rangle_d .$$

L'automate de la figure 2.13 remplit cette condition.

Sans surprise, un π-AFD sur lequel la propriété *Trace-DC* est vérifiée admet une projection de contrôle consistante (propriété OCC, section 2.3). Il n'existe cependant pas de procédure efficace pour trouver de tels π-AFD ; on doit en général explorer tout l'espace des solutions.

La propriété *Trace-DC* ne permet pas à elle seule de garantir la *consistance hiérarchique* (section 2.3) d'un π-AFD. L'AFD de la figure 2.13 en illustre les raisons. Pour inhiber une transition dans $\mathbf{G}^\pi$, il faut trouver une configuration de contrôle dans $\mathbf{G}$ dont l'application procure l'effet voulu. Sur la figure on voit qu'il suffit d'inhiber σ_2 dans $\mathbf{G}$ pour que τ_1^2 soit inhibé dans $\mathbf{G}^\pi$. De plus l'inhibition de σ_2 dans $\mathbf{G}$ n'a aucun effet sur τ_1^3 ni sur τ_1^4 ; ces deux événements peuvent toujours se produire dans $\mathbf{G}^\pi$. Mais pour inhiber τ_1^3 dans $\mathbf{G}^\pi$, on n'a pas d'autres choix que d'inhiber σ_1 dans $\mathbf{G}$; c'est la seule configuration de contrôle ayant l'effet requis. Or l'inhibition de σ_1 dans $\mathbf{G}$ provoque aussi (implicitement) l'inhibition de τ_1^2 dans $\mathbf{G}^\pi$. Ainsi $\mathbf{G}$ n'est pas en mesure de produire tous les sous-langages contrôlables que l'on peut sélectionner sur $\mathbf{G}^\pi$.

Pourvu que $\mathbf{G}^\pi$ soit *Trace-DC*, on peut vérifier que chaque cellule $\mathcal{G}_i$ (de $\mathbf{G}$) est en mesure d'implémenter, de façon indépendante (sans inhibition implicite), toutes les configurations de contrôle que l'on peut spécifier sur Q_i dans $\mathbf{G}^\pi$. Si cette dernière condition est remplie, on dit que $\mathbf{G}^\pi$ est *Non-Blocking IBC*, et alors $\mathbf{G}^\pi$ est *hiérarchiquement consistant*.

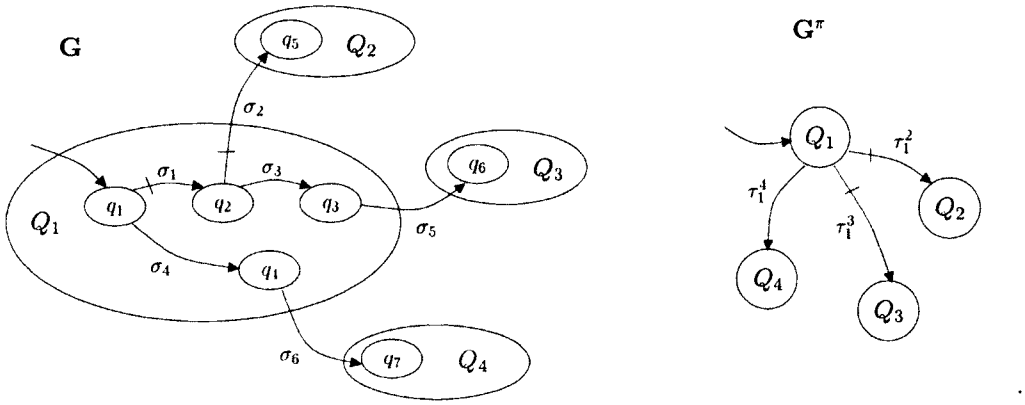


FIGURE 2.13 – π-AFD *Trace-DC*

Sans compter que la vérification de la propriété *Non-Blocking IBC* est coûteuse, cette propriété n'est pas stable sous la conjonction et la disjonction de partitions dans le treillis des partitions sur Q. Il y a donc peu d'espoir de trouver des procédures efficaces nous permettant de synthétiser des partitions possédant les propriétés requises à partir d'une approximation initiale (par analogie aux procédures de synthèse de SCT).

Génération de contrôleurs SCT

Chapitre 3

Composantes réutilisables pour SCT, position du problème et éléments de solution

La revue du domaine présentée dans les chapitres 1 et 2 révèle que la théorie du contrôle supervisé (SCT), bien qu'homogène dans l'ensemble de ses concepts, n'est pas homogène au niveau de sa présentation théorique. Les concepts fondamentaux dus à Ramadge et Wonham [43, 44, 45] (souvent appelé le *contexte usuel* de SCT) demeurent les mêmes mais il existe plusieurs variantes de la théorie dont certaines, basées sur un principe architectural de hiérarchie entre les composantes d'un système, sont qualifiées de *variantes hiérarchiques*. Parmi ces *variantes hiérarchiques* on retrouve notamment :

- le *contrôle hiérarchique* de Wong et Wonham [55, 58, 59, 60] ;
- le *contrôle hiérarchique par interface* (HISC) de Leduc, Brandin, Lawford et Wonham [35, 38] ;
- le *contrôle de STS* de Ma et Wonham [39].

On peut également y inclure la notion d'*automate partition* de Hubard et Caines [29] présentée à la section 2.8.2. Comme la théorie originale (de Ramadge et Wonham), ces variantes hiérarchiques possèdent toutes une notion explicite ou implicite de *composante* mais, mis à part la théorie du contrôle hiérarchique par interface (HISC), l'emphase n'est pas mise en général sur la notion de *composantes réutilisables*.

Le but de cette thèse est précisément d'étudier l'usage de composantes réutilisables dans la conception et l'implémentation de contrôleurs dérivés de SCT et à cette fin une analyse des variantes hiérarchiques de SCT semble s'imposer. L'analyse doit tenter de déterminer si l'une de ces variantes peut être utilisée intégralement avec une forme de composantes réutilisables. Sinon on doit déterminer s'il a un moyen de combiner ensemble certains éléments en provenance de plusieurs de ces variantes pour former une méthode qui permette la formulation et la résolution de problèmes de contrôle. L'analyse doit cependant tenir compte de tous les aspects du problème, de la modélisation en passant par la synthèse et en incluant l'implémentation automatique de la solution. L'objectif est ambitieux et il importe d'en réduire sensiblement l'ampleur.

D'emblée on peut considérer que le contexte usuel de SCT (celui de Ramadge et Wonham) se prête mal à l'usage de composantes réutilisables dans la formulation de problèmes de contrôle. Dans ce paradigme les modèles utilisés sont des modèles de sous-systèmes et non pas, à proprement parler, de composantes ; les modèles ne sont pas génériques. On peut contempler en réutiliser la *structure* mais pas directement la définition. De plus ces modèles sont aussi élaborés *en fonction* du problème de contrôle considéré, et sortent complètement du contexte théorique de SCT, leur élaboration se fait donc par un processus *ad hoc*. La synthèse modulaire (section 2.1) n'apporte rien de nouveau et demeure soumise aux mêmes limitations. On peut donc les éliminer comme *candidats* potentiel de solution.

Quant au paradigme du contrôle sous observation partielle (section 2.2) et aux variantes du *contrôle distribué* formulées dans le contexte de l'observation partielle (voir [32]), l'impossibilité d'une synthèse non bloquante les rendent également assez peu intéressants. On peut donc aussi les exclure de l'analyse. Remarquons cependant que l'observation partielle est nécessaire si on entend utiliser une forme d'*encapsulation* pour définir la métaphore de composante réutilisable. Heureusement les variantes hiérarchiques utilisent toutes *implicitement* une forme d'observation partielle, bien qu'elle ne soit pas formellement définie. Seul le contrôle hiérarchique aborde cette question (voir Wong [55] en appendice B) pour prouver que la notion d'observateur (section 2.3.3) est équivalente à l'observation partielle.

La variante du contrôle par prédicats (section 2.5) n'est pas une théorie complète en ce sens qu'elle n'est pas assortie de procédures de synthèse dans [62]. Zhang [63] élabore des procédures appropriées pour ce type de contrôle, mais le contexte de modélisation présumé

est alors le contexte usuel de SCT. En fait le contrôle par prédicats trouve son expression la plus intéressante dans la théorie du contrôle de STS de Ma et Wonham. Il n'est donc pas nécessaire d'en faire une analyse indépendante.

En dernier lieu, on peut considérer que les travaux de Wong et Wonham sur le contrôle hiérarchique *couvrent* la notion d'*automate partition* de Hubbard et Caines à travers la notion d'observateur (voir à ce sujet [60]). Or le support formel et algorithmique pour l'élaboration d'observateurs résultant en un quotient de l'AFD du modèle observé (précisément un type d'*automate partition*) est plus précis et plus complet. La notion d'*automate partition* de Hubbard et Caines peut donc aussi être exclue de l'analyse.

L'analyse peut donc se confiner à l'ensemble des variantes hiérarchiques de SCT, puisqu'elles semblent présenter les caractéristiques les plus adéquates par rapport à l'objectif global de cette étude.

3.1 Position du problème et grille d'analyse

Le problème général de conception et d'implémentation de contrôleurs dérivés de SCT peut être divisé en trois sous-problèmes principaux :

- le problème de la modélisation ;
- le problème de la synthèse ;
- le problème de la génération de code.

Le sous-problème de la *modélisation* se divise lui-même en deux sous-problèmes :

- la modélisation du système à contrôler (comportement libre du système) ;
- la modélisation du processus à implémenter sur ce système (les *contraintes* ou *requis* du contrôle).

Le but de cette étude est d'identifier une métaphore de composantes réutilisables permettant de supporter l'automatisation (partielle ou complète) de la solution à chacun de ces problèmes ; il faut donc préciser cette notion de *composante réutilisable*.

3.1.1 Notion intuitive de composante réutilisable

L'utilité et la nouveauté de SCT réside largement dans la possibilité d'une *synthèse* de la logique de contrôle. Le problème de la synthèse pose à son tour la question de la disponibilité de procédures de synthèse. Seules les variantes hiérarchiques de SCT (en plus du contexte usuel de SCT) définissent des procédures de synthèse, et toutes reposent sur des descriptions du système sous forme de structures de transition. La modélisation doit donc utiliser des structures de transition (des AFD ou des *holons*, section 2.7.1.2). Or, comme le fait remarquer Leduc dans [33], il est généralement nécessaire d'*élaborer* ces représentations car la dynamique des dispositifs utilisés dans les problèmes de contrôle n'est généralement pas décrite de façon aussi formelle dans les manuels techniques. Ce processus s'apparente à une forme d'*abstraction* des dispositifs physiques, dont le produit est un structure de transition (généralement un AFD) faisant office d'*interface* entre le dispositif physique et la spécification SCT.

3.1.1.1 Notion de composante dans le contexte usuel de SCT

Dans le contexte usuel de SCT (celui de Ramadge et Wonham) il existe déjà une notion intuitive de composante (section 2.1). Le comportement libre d'un système $\mathbf{G}$ peut prendre la forme d'une composition synchrone (composition parallèle) d'AFD $\mathbf{G}_i, i \in \{1, \ldots, n\}$:

$$\mathbf{G} := \|_{i \leq n} \mathbf{G}_i. \tag{3.1}$$

Cette expression algébrique exprime l'intuition selon laquelle le système $\mathbf{G}$ est constitué de *composantes* en fonctionnement parallèle concurrent. Cette notion intuitive de *composante* ne suffit-elle pas ?

Tout d'abord il devrait être clair que l'ensemble des modèles $\mathbf{G}_i$ de l'équation 3.1 ne correspondent pas à la notion de composante, telle qu'on entend généralement en génie logiciel, mais bien à la notion d'instance de sous-systèmes particuliers. Pour correspondre à la notion de composante il lui manque une notion adéquate d'*instanciation*. Dans la situation envisagée, il s'agit de modèles *génériques* de générateurs (voir section 1.1.3) dont l'ensemble des $\mathbf{G}_i$ constituent alors des *instances*. Or si les instances de composantes peuvent

partager des événements, ou s'il existe toute autre forme de dépendance entre elles, la notion d'instanciation devient délicate. Que veut dire alors d'*instancier* un générateur (que signifie G_i au juste) ? Les événements sont-ils *partagés* entre deux instances d'un même générateur *générique* (une composante réutilisable) ? Mais alors qu'en est-il des événements que partagent deux générateurs *génériques* distincts (deux composantes réutilisables distinctes) ?

Si on considère que deux instances d'une même composante réutilisable modélisent deux dispositifs physiques distincts au sein d'un système, il serait naturel d'imaginer que ces deux modèles possèdent chacun leur alphabet d'événements et que ces deux alphabets soient disjoints. Autrement dit deux instances d'une même composante réutilisable définissent des alphabets disjoints, des espaces d'états disjoints et des fonctions de transition disjointes, bien que les structures de transition soient *isomorphes* (aux étiquettes d'événements et d'états près). On parle alors de *composantes disjointes*.

Il est aussi difficile d'imaginer que plusieurs *composantes réutilisables* distinctes (dont les instances sont appelées à modéliser des dispositifs physiques distincts) puissent partager des événements, car alors ces composantes réutilisables ne peuvent jamais être instanciées séparément. On peut dire qu'un tel ensemble de composantes réutilisables constitue en fait un seul *module* qui décrit le comportement du matériel à l'aide de plusieurs générateurs partageant une partie ou la totalité de leurs événements. Par conséquent on a plutôt affaire à une seule *composante réutilisable* dont le modèle de comportement contient plusieurs générateurs dont les alphabets ne sont pas disjoints. Il est clair que dans ce dernier cas, puisqu'on doit toujours instancier l'ensemble des générateurs au complet lors d'une instanciation de la composante, on peut combiner les générateurs en question par *composition synchrone* pour obtenir une description unique du comportement de la composante réutilisable. On peut donc supposer aussi que les définitions de composantes réutilisables, prises deux à deux, définissent des alphabets disjoints, des espaces d'états disjoints et des fonctions de transition disjointes.

Une fois munies d'une notion d'instanciation appropriée, il devient possible d'utiliser des *composantes réutilisables* pour exprimer le comportement libre d'un système par simple composition synchrone d'instances de ces dernières comme à l'équation 3.1. Ce produit exprime le fait que les *sous-systèmes* (représentés par des *instances* de compo-

santes réutilisables) sont en fonctionnement parallèle concurrent. Puisque les alphabets des instances sont disjoints on obtient alors un *produit intercalé* (voir section 1.1.4). Mais pour formuler cette notion d'instanciation, on doit supposer que les instances sont toutes disjointes deux à deux. Cette limitation est-elle bien raisonnable ?

On constate que, dans la pratique, l'usage d'*usines modulaires* pour implémenter des procédés industriels est fort répandue. Ce type d'usine est un assemblage de *composantes physiques* modulaires (vérins, pinces, étaux entre autres) que l'on peut reconfigurer à volonté pour s'ajuster à différents *procédés de fabrication.* Chaque composante physique d'une usine modulaire dispose de ses propres capteurs et actionneurs. Les capteurs et les actionneurs sont associés à des entrée/sorties et sont à l'origine des *événements* perceptibles (ou générés) par la logique de contrôle. Comme les composantes physiques d'une usine modulaire ne partagent généralement pas leurs entrées/sorties, ces composantes physiques peuvent donc être considérées comme *disjointes* et en fonctionnement parallèle concurrent. Puisqu'il y a souvent beaucoup de redondance parmi les dispositifs utilisés pour constituer un système dans ce type d'usine (par exemple plusieurs vérins pneumatiques identiques quant à leur fonctionnement), l'usage de composantes réutilisables stéréotypées peut représenter une amélioration substantielle pour la modélisation de systèmes. Il est donc raisonnable de supposer dans cette analyse que les systèmes sont réalisés à partir d'instances disjointes de composantes réutilisables modélisant des sous-systèmes, eux-mêmes disjoints, comme dans le cas des usines modulaires.

La notion intuitive de *module* (c'est-à-dire les $\mathbf{G}_i$) implicite à l'équation 3.1 est souvent utilisée dans le contexte usuel de SCT. Or cette notion, bien que commode, souligne plusieurs difficultés relatives à la conception de modèles en SCT, surtout si l'on contemple l'usage de composantes réutilisables telles qu'esquissées dans ce qui précède. Entre autres,

- l'élaboration de composantes se fait de façon *ad hoc* ;
- les *modules* sont conçues en fonction d'un problème de contrôle spécifique.

Dans le contexte usuel de SCT, le processus d'élaboration des modèles n'est ni spécifié ni supporté de façon formelle par la théorie. Il s'agit donc d'un processus *ad hoc* complètement ouvert qui pose la question de la validité des modèles obtenus et, par conséquent, met aussi en question la qualité de la synthèse. On élabore généralement les modèles, correspondant aux différents *modules*, en fonction du problème de contrôle envisagé, de façon

à en réduire la complexité et à faciliter l'élaboration de *contraintes* (expression des requis du contrôle). On cherche aussi, en général, à concevoir ces modèles de sorte à éviter le problème de l'explosion de l'espace des états (voir au chapitre 2) qui rend la synthèse impossible à cause de l'espace requis pour les calculs. En conséquence le potentiel de réutilisation des modèles obtenus pour ces *modules* s'en trouve sévèrement compromis.

Par opposition, une composante réutilisable ne doit reposer sur aucune hypothèse a priori concernant le problème de contrôle à résoudre. Idéalement elle doit être utilisable dans n'importe quel problème de contrôle où l'ensemble des services qu'elle offre s'avère adéquat. Sur une telle base, l'élaboration d'une composante réutilisable doit viser à exposer *tous les services* que le matériel modélisé est en mesure de procurer. Il s'agit donc d'un modèle exhaustif détaillé de la dynamique du dispositif modélisé. Ce type de modèle est mal adapté à un usage correspondant à la notion intuitive de composante de SCT. Dans le cas d'un problème de contrôle particulier, un modèle exhaustif se révèle souvent *trop riche*, c'est-à-dire que le problème de contrôle n'utilise en général qu'un petit sous-ensemble du répertoire des actions que peut réaliser le dispositif. Le nombre de ces possibilités peut être élevé, surtout s'il s'agit d'un dispositif programmable, même si on programme ce dernier dans le but d'un usage spécialisé. En conséquence la structure de transition du modèle est souvent inutilement volumineuse ce qui complique l'expression des requis de contrôle et expose l'utilisateur au problème d'explosion de l'espace des états.

3.1.1.2 Problème de la conception *ad hoc* de modèles

Les problèmes relatifs à la conception de composantes sont rarement abordés dans la littérature concernant SCT, probablement à cause de la nature *ad hoc* du processus. Dans le contexte usuel on suppose implicitement que ce travail de modélisation est simplement répété pour chaque problème de contrôle. Or pour obtenir un modèle adéquat on doit souvent faire nombre d'hypothèses quant au comportement du matériel utilisé (des hypothèses spécifiques) et souvent valider ces hypothèses de façon statistique sur le matériel lui-même ou tout au moins par simulation. Ce travail minutieux, sujet aux erreurs, fait habituellement l'objet de plusieurs révisions avant d'en arriver à un modèle satisfaisant. Mais il est tout aussi essentiel qu'inévitable puisque la qualité de la synthèse en dépend. Ainsi l'élaboration d'un modèle adéquat de façon *ad hoc*, bien qu'inévitable, est un processus très laborieux

qui gagnerait à être *réutilisable*.

Pour mieux situer le problème il est utile d'en faire l'illustration sur un exemple simple de dispositif fréquemment rencontré dans les usines modulaires. Cette illustration procure aussi l'occasion d'initier le lecteur aux conventions de notations et de nomenclature ainsi qu'à l'interprétation des éléments des diagrammes utilisées dans les sections subséquentes. On y propose donc une méthode *ad hoc* pour la conception de *modèles exhaustifs* de composantes qui se veut typique dans son genre.

La figure 3.1 illustre la conception du modèle exhaustif du comportement d'un *vérin pneumatique à commande monostable*. On distingue clairement trois phases.

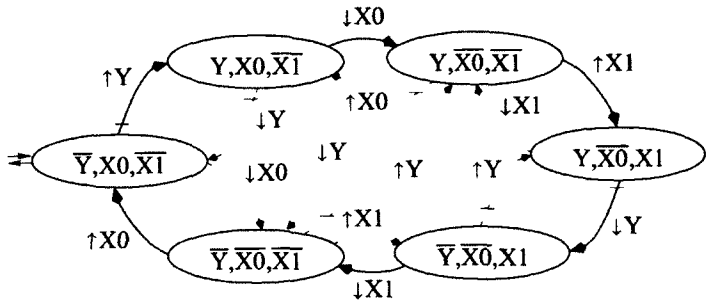
1. D'abord un *modèle brut* est établi (figure 3.1 (a)).
2. Ensuite on dégage le *comportement désiré* (figure 3.1 (b)) qui doit éventuellement être obtenu par programmation et qu'on appelle *pré-conditionnement* du dispositif.
3. Enfin on dégage un modèle exhaustif final (figure 3.1 (c)) où on change éventuellement les étiquettes d'événement ou d'état.

La phase d'établissement du *modèle brut* vise à identifier clairement l'espace des états que peut prendre le dispositif ainsi que l'ensemble des événements qu'il génère. Cette tâche doit s'appuyer sur les éléments qui sont perceptibles à la logique de contrôle, notamment l'état des capteurs et des actionneurs (respectivement, les entrées et les sorties du dispositif). On suppose ici que l'élément de contrôle est un automate programmable.

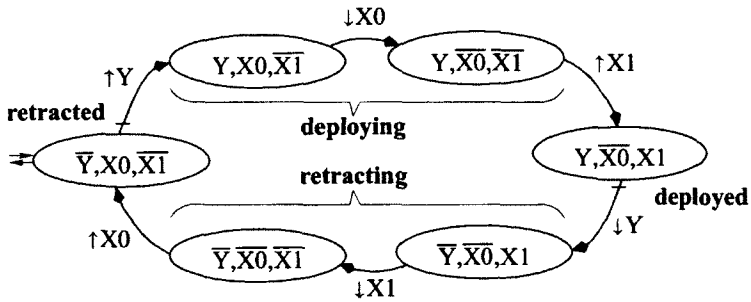
Sur la figure 3.1 (a), chaque état est assorti d'une *signature* prenant la forme d'un vecteur de formules logiques, chaque formule associée à une variable. Ces variables correspondent aux actionneurs et aux capteurs du dispositif et définissent un ensemble ordonné interprété comme un *vecteur d'état* (<Y,X0,X1> dans la figure). Par convention on utilise des préfixes pour distinguer les capteurs des actionneurs.

1. Le préfixe «Y» désigne un actionneur binaire.
2. Le préfixe «X» désigne un capteur binaire.

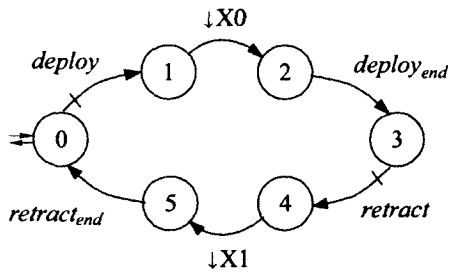
Lorsque plus d'un élément du même type est requis, on y ajoute un index, par exemple X0 et X1 dans la figure pour les capteurs de fin de course du vérin, et Y pour sa commande. Les *signatures* s'interprètent comme une formule logique, par exemple l'état $<\overline{Y},X0,\overline{X1}>$ vérifie la formule $F_0 = \overline{Y} \wedge X0 \wedge \overline{X1}$. L'utilisation du vecteur d'état peut être utile pour



(a) Modèle brut du matériel



(b) Modèle du matériel conditionné par programmation



(c) Modèle exhaustif final

FIGURE 3.1 – Modélisation *ad hoc* d'un vérin à commande monostable

détecter que l'espace d'états défini par les entrées/sorties n'est pas suffisant pour exprimer tous les états que peut prendre le dispositif. Les événements que peut générer le dispositif sont liés à des changements du vecteur d'état, c'est-à-dire qu'il s'agit principalement de fronts montants et descendants des variables du vecteur d'état.

L'établissement d'un comportement désirable (phase (2)) vise à identifier l'ensemble des services que doit procurer le dispositif et les modalités de ces services (par exemple si oui ou non une requête de service peut résulter en un échec). On vise en particulier à éliminer toute instabilité dans le modèle de la figure 3.1 (a) en éliminant l'effet de l'inertie dans le dispositif résultant d'un usage chaotique de la commande.

Dans la figure 3.1 (a), le *modèle brut*, certaines transitions sont identifiées en pointillé. Ces transitions sont liées à un *usage chaotique* de la commande du vérin c'est-à-dire que si la commande n'est pas d'avance soumise à un certain schéma d'usage ordonné, toutes ces transitions sont possibles dues à l'inertie du vérin. L'effet de l'inertie doit généralement être éliminé par programmation (pré-conditionnement du *modèle brut*) et par validation du modèle résultant car, par la définition de la contrôlabilité, toutes les transitions sur événements incontrôlables du *modèle brut* sont préservées par la synthèse SCT. Alors, même s'il était possible d'éliminer l'effet de l'inertie par calcul d'une loi de contrôle correspondant à une synthèse SCT, le modèle théorique du système en boucle fermée (voir section 1.3.2) résultant de la synthèse contiendrait toujours ces transitions parasites et ne serait donc pas fidèle à la réalité (puisque l'effet de l'inertie a justement été éliminé).

L'élaboration du modèle exhaustif final (phase (3)) ne vise qu'à rendre le modèle plus expressif, par exemple en changeant l'étiquette des événements, ou à en rendre l'expression plus économique, par exemple en remplaçant les *signatures* des états par des numéros. Il ne s'agit que d'un réétiquetage, on forme donc des *alias*. On a alors que l'événement $deploy$ devient équivalent à la commande de déploiement du vérin qui consiste à causer l'occurrence de l'événement $\uparrow$Y, que $deploy_{end}$ correspond à l'occurrence d'un front montant du capteur X1 ($\uparrow$X1) ou encore que l'état «0» correspond à la formule logique $F_0 = \overline{\mathrm{Y}} \wedge \mathrm{X0} \wedge \overline{\mathrm{X1}}$.

3.1.1.3 Solution possible au problème de conception *ad hoc*

En apparence il semble y avoir conflit entre les nécessités relatives à la modélisation et à la synthèse de problèmes de contrôle, qui requièrent des *modèles compacts* fiables, et la nature laborieuse et aléatoire de la confection de composantes selon le modèle intuitif de SCT. Le manque de *support formel* pour la confection de composantes semble particulièrement épineux. Or, comme mentionné par Leduc [33], il est clairement impossible d'éviter d'utiliser un processus *ad hoc* lorsqu'il s'agit de décrire le comportement d'un dispositif matériel de façon formelle. Leduc apparente ce processus à l'*abstraction* d'un genre d'*interface formelle* compatible à l'usage dans les procédures de synthèse. Une mesure d'arbitraire semble donc inévitable en ce qui concerne l'élaboration de modèles exhaustifs à partir de dispositifs matériels.

Dans la perspective qui précède, l'élaboration de composantes réutilisables peut être envisagée en deux phases.

1. Pour tout dispositif matériel, on doit d'abord procéder à l'élaboration d'un modèle *exhaustif* de son comportement par un processus *ad hoc*.
2. On peut ensuite envisager d'*abstraire* du modèle exhaustif un ou plusieurs modèles, plus compacts et mieux adaptés à un problème de contrôle particulier ou a un classe de problèmes de contrôle.

La première phase (*ad hoc*) vise à élaborer une représentation formelle exhaustive du comportement d'un dispositif matériel (par exemple un AFD, comme dans [33]). Ce modèle décrit en détail *toutes les possibilités offertes* par le dispositif modélisé (l'ensemble de ses services), et peut éventuellement être utilisé directement dans la modélisation d'un problème de contrôle. Il s'agit du *modèle exhaustif* ou *modèle détaillé* de la composante. La deuxième phase vise à *abstraire* un *modèle d'interface* simplifié par *encapsulation* des éléments de la dynamique détaillée du modèle exhaustif ne présentant pas d'intérêt pour certains problèmes de contrôle. Le modèle d'interface doit naturellement être utilisable dans la modélisation de problèmes de contrôle, donc il s'agit aussi d'une structure de transition appropriée. Le même dispositif de base (modèle exhaustif) est ainsi réutilisé sous plusieurs *interfaces*, et la composante éventuellement réutilisée par instanciation dans plusieurs problèmes de contrôle. La possibilité d'utiliser un support formel pour la modélisation de

composantes apparaît donc plus clairement dans la deuxième phase, puisqu'alors on dispose déjà d'un modèle exhaustif formel du dispositif à contrôler. L'usage d'un support formel pour l'abstraction d'interfaces présente l'avantage de confiner l'élément d'arbitraire, inévitable dans l'abstraction du comportement d'un dispositif physique, à l'élaboration du modèle exhaustif. Toutes les interfaces dérivées de ce modèle exhaustif jouissent alors de certaines garanties formelles, ce qui limite l'incertitude et permet d'amortir l'effort investi dans le modèle exhaustif.

Dans ce genre de processus d'élaboration de composantes en deux phases (*modèle + abstraction*), on peut même envisager que la phase d'abstraction puisse être répétée le long de l'*axe vertical*. Par exemple, on peut associer un ensemble de composantes (vérins et ventouse pneumatique) en un sous-système formant une *grue*, pour ensuite abstraire le comportement de cette grue en une interface appropriée plus *naturelle* à utiliser dans un problème de contrôle impliquant l'usage d'une grue. Une telle manoeuvre implique un genre de *synthèse partielle* sur le sous-ensemble des composantes utilisées pour obtenir une *loi de contrôle locale* correspondant au comportement d'une grue. Cette loi de contrôle est obtenue par application d'un ensemble de requis de contrôle dont la portée est purement locale, ayant préséance sur tout autre requis de contrôle en provenance du système global. Le système global contrôle donc une *grue* et non pas un ensemble amorphe constitué de vérins et d'une ventouse pneumatique. Ce type d'abstraction situe les requis de contrôle le plus près possible de l'endroit où ils s'appliquent (un type de modularisation des requis de contrôle) et procure à la conception une perspective plus significative des dispositifs et des sous-systèmes à contrôler (ce qui peut aider dans la formulation des requis).

3.1.2 Composantes réutilisables en modélisation et en synthèse

Pour guider l'analyse et se donner une idée des éléments à rechercher dans la littérature sur SCT, il peut être utile de se donner une idée approximative de la forme que peut prendre l'usage de composantes réutilisables dans la modélisation de problèmes de contrôle. Ce qui suit n'a que valeur d'illustration. On ne considère d'abord que les sous-problèmes de modélisation et de synthèse.

3.1.2.1 Scénario d'usage de composantes réutilisables

Pour poursuivre l'argumentation de la section 3.1.1, une composante réutilisable peut être modélisée par une paire de générateurs $\mathbf{C} := (\mathbf{G}, \mathcal{G})$, où $\mathbf{G}$ représente le générateur du modèle exhaustif de la composante et $\mathcal{G}$ le générateur de son interface abstraite. Une composante réutilisable $\mathbf{C}$ est ainsi un *stéréotype* correspondant à une *classe* de dispositifs partageant les mêmes caractéristiques de fonctionnement et à ce titre les générateurs $\mathbf{G}$ et $\mathcal{G}$ sont considérés *génériques*. On peut dire que $\mathbf{C}$ correspond à la notion de *classe* en programmation par objets. En poursuivant cette analogie on peut considérer qu'une instance de $\mathbf{C}$, nommée C, constitue un *espace symbolique* (l'équivalent de la notion de *namespace* en anglais) où sont définis $\mathbf{G}$ et $\mathcal{G}$. En programmation par objets, l'analogie correspondante est celle de la déclaration d'une variable *nommée* C dont le *type* est $\mathbf{C}$; une *classe*. L'effet d'une telle déclaration, au niveau de l'espace symbolique où cette déclaration se situe, est de redéfinir implicitement tous les symboles compris dans la définition de la *classe* $\mathbf{C}$. La *désignation* des symboles de $\mathbf{C}$ devient alors *relative* à la variable C.

Dans le cas qui nous intéresse, pour une instance C de $\mathbf{C} = (\mathbf{G}, \mathcal{G})$, on a alors deux générateurs $C.\mathbf{G}$ et $C.\mathcal{G}$ où l'usage du préfixe «$C.$» correspond à l'usage des noms qualifiés répandu en programmation par objets, et spécifie un *espace symbolique*. Ainsi on peut considérer que le modèle d'un système est composé d'un ensemble de variables C_i pour $i \in \{1, ..., n\}$, où chaque variable est associée à une composante réutilisable $\mathbf{C}_\alpha := (\mathbf{G}_\alpha, \mathcal{G}_\alpha)$ ($\alpha \in A$, un index) qui en définit le *type* (ou stéréotype). Bien sûr pour C_i, C_j ($i \neq j$) le stéréotype utilisé $\mathbf{C}_\alpha$ peut être le même, mais puisque C_i et C_j définissent des *espaces symboliques* disjoints et que chacun redéfinit *tous les éléments symboliques* de la composante réutilisable $\mathbf{C}_\alpha$ (la *classe*) y compris leurs alphabets, alors les alphabets de $C_i.\mathbf{G}$ et $C_j.\mathbf{G}$ sont disjoints ainsi que ceux de $C_i.\mathcal{G}$ et $C_j.\mathcal{G}$ de sorte que C_i et C_j représentent des *composantes* disjointes (au sens de *sous-systèmes* comme pour le contexte usuel de SCT).

Dans l'expression d'un problème de contrôle, le modèle du système est donc constitué de *variables typées*, C_i pour $i \in \{1, ..., n\}$, correspondant à des *instances* de composantes réutilisables ($\mathbf{C}_\alpha$ pour un index $\alpha \in A$). Chaque variable est associée à une composante réutilisable (son *type*) de façon indépendante. L'instanciation n'est pas *effective* au moment de la modélisation mais elle y est *implicite* dans le sens que chacune de ces variables est

éventuellement *assignée*, à l'exécution, à un et un seul dispositif qu'elle modélise. On obtient donc un ensemble de paires de générateurs $(\mathbf{G}_i, \mathcal{G}_i)$, où $\mathbf{G}_i := C_i.\mathbf{G}$ et $\mathcal{G}_i := C_i.\mathcal{G}$, avec l'assurance que l'ensemble des $\mathbf{G}_i$ ont des alphabets disjoints deux à deux ainsi que l'ensemble des $\mathcal{G}_i$, c'est-à-dire que les modèles des sous-systèmes sont disjoints. Globalement on obtient aussi deux modèles pour le système, soit le générateur

$$\mathbf{G} := \|_{i \leq n} \mathbf{G}_i$$

qui modélise le comportement libre du système de façon exhaustive, et le générateur

$$\mathcal{G} := \|_{i \leq n} \mathcal{G}_i$$

qui modélise en quelque sorte son comportement libre *abstrait*.

On remarque ici que le problème de l'instanciation n'a pas d'impact direct sur la formulation du problème de contrôle, en ce sens que la formulation du comportement libre du système ne change pas, pourvu que l'hypothèse initiale de sous-systèmes disjoints soit maintenue. Le problème d'instanciation se résume à procurer un mécanisme syntaxique adéquat garantissant de préserver cette hypothèse. Cependant l'usage de composantes réutilisables introduit deux modèles, le modèle *concret* $\mathbf{G}$ et le modèle *abstrait* $\mathcal{G}$. L'introduction d'un modèle *abstrait* dans la description du modèle d'un système est un changement significatif.

En général, on veut éviter de travailler directement sur le modèle concret $\mathbf{G}$ et plutôt utiliser le modèle abstrait $\mathcal{G}$ dont on espère la structure moins complexe et mieux adaptée au problème de contrôle à résoudre. Ainsi on veut pouvoir exprimer les requis du contrôle sur $\mathcal{G}$ et, à l'aide de procédures de synthèse adéquates, calculer une loi de contrôle sur $\mathbf{G}$ correspondant à ces requis et procurant des garanties similaires au contexte usuel de SCT (solution contrôlable et non bloquante au problème de contrôle). Or, pour que ce schéma puisse fonctionner, il faut pouvoir assurer un forme de propriété de *consistance* entre l'abstraction $\mathcal{G}$ et son modèle concret $\mathbf{G}$. Cette propriété de *consistance* doit assurer que, ou bien la synthèse effectue un calcul direct de la solution sur $\mathbf{G}$, ou bien la synthèse calcule une solution abstraite (une loi de contrôle sur $\mathcal{G}$) que l'on peut *reporter* sur $\mathbf{G}$ d'une

façon ou d'une autre.

On remarque que la phase d'abstraction du processus *modèle + abstraction* présuppose une forme d'*encapsulation*. En informatique on reconnaît deux principes intuitifs relatifs à l'encapsulation. On parle d'encapsulation *opaque* (de l'anglais *black-box encapsulation*) et d'encapsulation *ouverte* (*white-box encapsulation*). On peut, très grossièrement, classifier les variantes hiérarchiques de SCT selon ces notions intuitives. Le *contrôle hiérarchique* (voir section 2.3), à travers la notion de *projection causale*, procure une forme d'encapsulation *opaque*. Il en va de même pour HISC (section 2.4) à travers sa notion explicite et formelle d'interface. Dans ces dernières variantes l'encapsulation est *opaque* en ce sens que la dynamique détaillée des modèles exhaustifs (ou modèles concrets) est inaccessible aux procédures de synthèse, seule la structure des interfaces est visible au niveau abstrait. Dans ce cas la *consistance* entre le modèle abstrait et le modèle concret doit être assurée par un ensemble de propriétés vérifiables sur les interfaces, tandis que le *contrôle de STS* (section 2.7), à travers la notion de superposition de *holons*, procure une encapsulation *ouverte* en ce sens que la procédure de synthèse *utilise* la dynamique détaillée des modèles concrets (les *holons*). L'encapsulation est donc *ouverte* en ce sens que la dynamique des modèles concrets est accessible à la procédure de synthèse et que la solution calculée est applicable directement au modèle concret du procédé. Dans ce dernier cas l'outil de modélisation allié à des procédures de synthèse adaptées procure la *consistance* entre modèles abstrait et concret.

L'usage d'une encapsulation ouverte procure en général une solution de synthèse *optimale* selon les critères de SCT, c'est-à-dire que la solution correspond au plus grand sous-langage contrôlable et non bloquant de l'objectif de contrôle. Autrement dit toutes les options de contrôle demeurent accessibles à la synthèse. Par contre l'usage d'une encapsulation opaque résulte habituellement en une solution de synthèse non optimale puisque les procédures de synthèse n'ont alors pas accès aux options de contrôle encapsulées dans les composantes abstraites. En général, cette perte d'optimalité dans des solutions de synthèse n'est pas considérée critique pourvu que les solutions obtenues demeurent *adéquates* du point de vue pragmatique. Une synthèse optimale en SCT ne correspond pas nécessairement à une notion pragmatique du contrôle *optimal* d'un dispositif. Les critères pragmatiques sont en général plus variés, moins bien définis et puisqu'ils interagissent entre-eux

ils sont souvent de nature variable. Pour véritablement tenir compte de critères d'optimalité de nature pragmatique, il faudrait sans doute faire intervenir des fonctions de coût (voir à cet effet [28]). Il existe de nombreux cas où l'expression et l'encapsulation de requis de contrôle de portée locale (par exemple l'usage d'un vérin et d'une ventouse pneumatique encapsulés pour procurer le comportement d'une grue) clarifient l'expression d'un problème de contrôle et de ses requis au niveau global. Une encapsulation opaque du type offert par le contrôle hiérarchique peut même réduire significativement la complexité de la synthèse, entre autres avantages. Aussi, lorsque l'encapsulation opaque procure des bénéfices appréciables sans trop hypothéquer la synthèse, il peut être admissible de l'utiliser. Mais c'est là, bien sûr, où intervient le jugement d'un concepteur de composantes réutilisables.

3.1.2.2 Critères d'analyse liés à la modélisation et à la synthèse

De ce qui précède on peut déjà dégager un ensemble critères utiles pour la recherche d'une métaphore de composantes réutilisables.

C1. On vise à identifier les éléments de modularisation permettant de supporter une métaphore de composante réutilisable du type *modèle + abstraction.*

C2. La méthode d'abstraction peut utiliser une forme d'encapsulation opaque ou ouverte, mais elle doit procurer une forme de *consistance* afin de garantir qu'une synthèse contrôlable et non bloquante sur le modèle abstrait puisse être implémentée sur le modèle concret (par une loi de contrôle appropriée).

C3. On vise à identifier la possibilité de disposer d'un support formel pour la dérivation d'interfaces (l'*abstraction*).

C4. Il est désirable que le processus d'abstraction puisse être répété sur l'axe vertical de manière à *encapsuler* des requis de contrôle de portée locale.

C5. On vise à identifier la disponibilité de procédures de synthèse adaptées (ou du moins adaptables) à l'usage de composantes réutilisables.

C6. On vise à examiner la possibilité d'utiliser les méthodes symboliques (section 2.6) dans les calculs de synthèse.

L'usage des méthodes symboliques pour la synthèse n'est pas un préalable à l'usage (ou à la définition) d'une métaphore de composantes réutilisables, mais il peut s'avérer utile pour considérer la synthèse sur des systèmes d'envergure réaliste (voir Zhang et Wonham [64] et Ma et Wonham [39]).

Aussi, les limitations suivantes semblent raisonnables et utiles pour l'analyse.

L1. On peut considérer a priori que les composantes d'un système (au sens de *sous-systèmes* comme dans le contexte usuel de SCT) sont disjointes (cas des usines modulaires).

L2. On peut considérer des formes d'encapsulation opaques même si cela entraîne a priori une certaine perte d'optimalité de la synthèse (selon les critères de SCT).

L3. On peut se limiter à des solutions de synthèse *monolithiques*, c'est-à-dire que l'architecture de contrôle est centralisée.

Une fois de plus la synthèse monolithique (par opposition à la synthèse modulaire ou distribuée, voir la section 2.1 par exemple) n'est pas un préalable à l'usage (ou à la définition) d'une métaphore de composantes réutilisables, mais cette limitation semble raisonnable vu l'envergure déjà considérable de la tâche.

3.1.3 Composantes réutilisables et implémentation de contrôleurs

L'un des buts de la présente thèse est d'examiner la possibilité d'une génération automatique de l'implémentation de contrôleurs issus d'une conception utilisant des composantes réutilisables en conjonction avec les procédures de synthèse de SCT. Cet aspect du problème impose des limitations strictes sur l'ensemble des solutions possibles qu'il faut garder à l'esprit même à ce stade préliminaire où l'on tente de définir une métaphore de composantes réutilisables pour SCT.

3.1.3.1 Problèmes liés à l'implémentation de contrôleurs

Les procédures de synthèses définies à ce jour produisent soit un *superviseur adéquat* (voir la section 1.3.2), la version formelle d'un contrôleur, soit une loi de contrôle de type SFBC (voir la section 2.5 pour la définition et [39], [52] pour les procédures de synthèse).

Un *superviseur adéquat* prend généralement la forme de l'AFD correspondant au comportement de la boucle fermée du système sous contrôle. Cette structure de transition peut être considérée en elle-même comme un contrôleur (figure 1.1). Ainsi une forme naïve d'implémentation consiste à implémenter la structure de transition d'un *superviseur adéquat* telle quelle (voir [11] entre autres). Cependant cette solution n'est en général pas praticable étant donnée la taille de cette structure de transition même pour des problèmes de contrôle relativement modestes. Si on contemple l'implémentation sur un automate programmable (PLC-*Programmable Logic Controller*), une solution fréquente en automatisation industrielle, cette solution devient alors tout à fait irréaliste.

Ainsi, l'implémentation de la structure de transition d'un *superviseur adéquat* est généralement hors de portée à cause de la taille de son espace d'états. Mais d'autres problèmes (détaillés dans [7] et aussi dans [33]) sont dus à la présence de parallélisme effectif entre dispositifs au sein de l'usine par rapport à la durée du cycle de rétroaction. Ces problèmes sont généralement causés par la nécessité de *calculer* l'état courant du système (c'est-à-dire du *superviseur adéquat*) à partir des événements en provenance de l'usine.

Le cycle de rétroaction se définit comme le temps nécessaire à effectuer le calcul de la rétroaction et se compose des éléments suivant :

- temps d'acquisition de l'état des capteurs avec mise à jour de l'image mémoire du procédé ;
- temps du calcul effectif de la rétroaction (c'est-à-dire de l'ensemble des commandes à émettre étant donné l'état du procédé) ;
- temps nécessaire à l'actualisation des actionneurs correspondant à ces commandes.

Pendant le calcul de la rétroaction, le procédé évolue et l'élément de contrôle (par exemple un automate programmable) n'est pas informé en temps réel de l'état du procédé. Dû à ce décalage temporel, puisqu'il y a parallélisme effectif entre les dispositifs en fonctionnement dans l'usine, plusieurs événements peuvent se produire avant le prochain cycle d'acquisition de l'état du procédé. Au prochain cycle de rétroaction, la logique de contrôle perçoit alors les occurrences d'événements comme *simultanés* et perd la trace de la *séquence d'occurrences* de ces événements. Ceci est une dérogation inévitable au modèle théorique qui peut entraîner que le contrôleur soit incapable de calculer l'état courant du système. Bien sûr, en supposant que le cycle de rétroaction soit très court, la probabilité

qu'une telle situation se présente peut sembler très faible. Cette hypothèse est cependant inadéquate puisqu'il y a parallélisme effectif des dispositifs élémentaires dans l'usine et ainsi le délai entre l'occurrence de deux événements se produisant dans deux dispositifs indépendants dans l'usine est arbitrairement court.

Par opposition, pour une loi de contrôle du type SFBC, bien que la rétroaction dépende aussi de l'état courant du système, celui-ci peut alors être modélisé comme un ensemble de dispositifs élémentaires correspondant aux dispositifs physiques de l'usine. Or au sein d'un dispositif élémentaire tel qu'un vérin par exemple, il n'y a pas de parallélisme. Les événements s'y produisent l'un après l'autre selon le modèle de comportement (AFD ou autre structure de transition) du dispositif. On peut donc en général suivre l'évolution du système (c'est-à-dire *calculer* son état global) de façon fiable en *calculant* l'évolution de chacun de ses dispositifs élémentaires. On peut réaliser ce *calcul* de deux façons.

Un premier scénario consiste à associer à chaque état des modèles des dispositifs élémentaires une *signature*, c'est-à-dire une formule logique basée sur l'état de ses entrées et sorties (capteurs, actionneurs et éventuellement aussi d'éléments de mémoire qui lui sont assignés en exclusivité). Chaque dispositif élémentaire dispose alors d'un *vecteur d'état* et l'évaluation de son état courant se fait par évaluation directe de formules logiques (voir [8]). Dans ce cas il n'y a pas de réserve quant à la durée du cycle de rétroaction. L'information d'état calculée est la meilleure que l'on puisse espérer en pareilles circonstances et, bien que l'évaluation des formules logiques puisse s'avérer onéreuse en temps, cette situation est comparable à une solution programmée de façon traditionnelle (par grafcets par exemple). La seule réserve concerne la génération automatique de l'implémentation qui peut s'avérer complexe et requérir beaucoup de support au niveau de la confection des modèles des dispositifs élémentaires à cause, entre autres, de la nécessité de déterminer les *signatures* logiques.

Un deuxième scénario consiste à tenir à jour un modèle mémoire interne (dans la mémoire de l'automate programmable) de l'évolution de chaque dispositif élémentaire et à *suivre* cette évolution à travers les événements perçus à chaque cycle d'acquisition (voir [52]). Dans ce cas la taille de la mémoire requise (dans l'automate programmable) n'est habituellement pas prohibitive puisqu'en général ces modèles sont assez simples. Aussi la seule hypothèse requise est que le cycle de rétroaction soit suffisamment court pour en-

registrer de façon fiable l'occurrence de chaque événement. Puisque qu'il n'y a pas de parallélisme et que le temps de réaction d'un dispositif physique est en général beaucoup plus long que le cycle de rétroaction d'un automate programmable, cette hypothèse devient alors raisonnable.

Ces deux scénarios d'implémentation sont viables mais il y a une difficulté supplémentaire. Contrairement à un *superviseur adéquat*, une SFBC telle que définie dans la section 2.5 n'a pas de *mémoire* c'est-à-dire que son contrôle ne peut pas varier selon l'historique de fonctionnement du procédé ; c'est une limitation sévère. Dans un système obtenu par produit intercalé de sous-systèmes indépendants, l'AFD de la boucle fermée du système sous le contrôle d'une SFBC est un *sous-automate* de l'AFD de comportement libre du système (voir [32], remarque 3.1). Par opposition, l'AFD d'un *superviseur adéquat* est souvent plus riche car il peut contenir plusieurs *instances* du fonctionnement de certains sous-systèmes. Bref, dans le cas général, un *superviseur adéquat* contient *implicitement* une *mémoire* intégrée qui lui permet par exemple de *compter* le nombre de fois qu'un certain sous-système a accompli une certaine tâche.

Ainsi, pour pouvoir implémenter un *superviseur adéquat* en suivant l'un des deux scénarios exposés précédemment, il faudrait disposer d'une représentation de sa mémoire. Pour la mémoire, puisqu'il s'agit de toute évidence d'un modèle interne à l'élément de contrôle (un PLC par exemple), l'implémentation devrait suivre le deuxième scénario. Or le problème de dériver automatiquement un modèle (par exemple un AFD) pour cette mémoire (par exemple à partir d'un *superviseur adéquat*) ne semble jamais avoir été abordé dans la littérature sur SCT. De toute façon ce problème est probablement NP-difficile puisqu'il implique nécessairement la minimisation du *superviseur adéquat* (de l'anglais *Supervisor Reduction*) qui est un problème que l'on sait être NP-difficile (voir [53]).

La situation n'est cependant pas complètement désespérée puisqu'il existe un type de SFBC, appelé DSFBC (de l'anglais pour *Dynamic State FeedBack Control*), qui incorpore une notion de mémoire (voir [62]) et qui permet de modéliser les mêmes problèmes de contrôle que ceux modélisés dans le contexte usuel de SCT. Le contrôle de STS de Ma et Wonham [39] est un exemple de synthèse de DSFBC où on modélise les éléments de mémoire de façon explicite dans le modèle du problème de contrôle en tant que *contraintes* (les requis du contrôle). Ainsi, en modélisant explicitement la mémoire dans le problème

de contrôle sous forme de *contraintes* et en utilisant une procédure de synthèse dont le résultat est une SFBC (donc on obtient une DSFBC), on peut envisager l'implémentation de la solution de contrôle par l'usage des deux scénarios illustrés précédemment. On modélise l'ensemble des dispositifs physiques élémentaires (l'*équipement*) à l'aide de l'un des deux scénarios, et on modélise la mémoire en utilisant le deuxième scénario. L'état courant combiné de l'*équipement* et de la mémoire détermine alors l'état courant du procédé.

3.1.3.2 Critères d'analyse liés à l'implémentation de contrôleurs

En plus des limitations énumérées à la section précédente on peut ajouter les suivantes qui semblent aussi raisonnables.

L4. On peut se limiter à des solutions de synthèse procurant une loi de contrôle de type SFBC.

L5. On peut supposer, pour les considérations de la génération de code, que la machine cible est un automate programmable.

L6. On exclut toute forme de calcul *en ligne* de la rétroaction, le contrôleur est synthétisé et calculé *hors ligne* (c'est-à-dire que le contrôleur est un programme *compilé* d'avance et téléchargé dans un automate programmable).

Les critères d'analyse suivants semblent aussi pertinents.

C7. On vise à identifier des procédures de synthèse dont le résultat permet de dériver une loi de contrôle de type SFBC.

C8. On doit porter une attention particulière au support requis pour générer automatiquement l'implémentation du contrôleur correspondant à la loi de contrôle obtenue par synthèse.

3.2 Contrôle de STS

La théorie du contrôle des structures en arborescence d'états (ou contrôle de STS), présentée en section 2.7 et détaillée dans [39], présente plusieurs caractéristiques qui la rendent attrayante comme alternative pour la modélisation de problèmes de contrôle et la synthèse

de leurs solutions suivant les concepts de SCT. La notation, similaire à celle utilisée pour les *statecharts* de Harel et Politi (voir [27]), est particulièrement claire et concise. Elle exprime élégamment les structures hiérarchiques dans les problèmes de contrôle ainsi que les relations de parallélisme. Il reste encore cependant à identifier s'il est possible de l'allier à une métaphore de composantes réutilisables.

3.2.1 Les avantages du contrôle de STS

La théorie s'articule autour d'un support de modularisation, le *holon* (section 2.7.1.2), permettant une forme d'encapsulation hiérarchique ouverte à travers son *principe de superposition de holons* (voir la définition 2.14 de [39], *match*). Le holon muni du principe de superposition (illustré en figure 2.11) offre donc un mécanisme formel d'abstraction.

Le holon est muni d'une définition formelle d'interface (figure 2.9 et figure 2.10). De plus le principe de superposition de holons est assorti d'un ensemble de règles de consistance à l'interface pour assurer la consistance de cette superposition par rapport à la dynamique des structures de transitions impliquées. On a donc une forme de consistance entre les modèles abstrait et concret quand à la dynamique des structures de transition.

La théorie fournit des procédures de synthèse qui utilisent la notion de superposition de holons pour calculer la relation de transition détaillée du modèle. La synthèse s'opère ainsi sur le modèle détaillé du problème de contrôle (encapsulation ouverte) procurant la consistance de la solution de contrôle entre les modèles abstrait et détaillé (ou concret).

Cette structure hiérarchique permet de définir un support formel pour la notion de *famille d'états* ; l'arborescence d'états. La théorie établit aussi la notion de projection $\Theta(\cdot)$ (section 2.7.3) qui procure un support formel pour obtenir une fonction caractéristique (un prédicat) représentant une arborescence d'états (donc une famille d'états). La définition formelle de $\Theta(\cdot)$ constitue en elle-même une procédure pour l'encodage d'une arborescence d'états sous forme de BDD pour la synthèse. Il s'agit donc d'une méthode de synthèse par prédicats dont le résultat est une loi de contrôle de type SFBC utilisant les méthodes symboliques (section 2.6).

L'usage des méthodes symboliques en synthèse réduit la possibilité de problèmes relatifs à l'explosion de l'espace des états. En conséquence la taille des problèmes de contrôle

que l'on peut aborder correspond à des situations plus réalistes (voir [63] et [39] entre autres). En plus la théorie fait usage d'une notion de contrôlabilité dite *faible*, moins exigeante que la propriété de contrôlabilité établie pour le modèle avec prédicats (dans [62]). En conséquence la complexité des calculs de synthèse s'en trouve significativement réduite (voir section 2.7.2).

Aussi, comme on peut le constater dans les exemples fournis dans [39] (en section 4, 5 et 6), le formalisme de modélisation favorise la modélisation *explicite* d'éléments de mémoire pour modéliser des contraintes dans les problèmes de contrôle. La spécification des requis de contrôle se fait alors en énumérant les familles d'états prohibées (les états du système global que la solution contrôlable doit éviter ; des prédicats). Cette forme de spécification offre une plus grande clarté par rapport à la forme utilisant des AFD pour spécifier un *langage légal* (forme utilisée dans le contexte usuel de SCT). Comparée à la spécification d'un *langage légal*, la spécification par évitement de familles d'états prohibés est beaucoup plus simple à utiliser et offre même la possibilité d'introduire une notion de traçabilité des requis de contrôle.

Finalement, la modélisation explicite d'éléments de mémoire pour modéliser des contraintes dans les problèmes de contrôle rend le contrôle de STS capable d'aborder une classe de problèmes de contrôle comparable à ce que le contexte usuel de SCT peut résoudre (voir [62]). La loi de contrôle obtenue par synthèse est en fait une SFBC *dynamique* (DSFBC). Le contrôle de STS offre ainsi le potentiel d'une implémentation élégante et fiable le long des principes discutés en section 3.1.3.

3.2.2 Composantes réutilisables et contrôle de STS

En contrôle de STS (section 2.7) il existe plusieurs unités de modularisation. Un STS $\mathrm{G} = (\mathrm{ST}, \mathcal{H}, \Sigma, \Delta, \mathrm{ST}_0, \mathrm{ST}_m)$ représente lui-même un module c'est-à-dire le *système*. Il est doté d'un *espace d'états* structuré en forme d'arbre, l'*arborescence d'états* $\mathrm{ST} = (X, x_0, \mathcal{T}, \mathcal{E})$, par l'usage de différents types d'états : des états simples et des états structurés (super-états OU et super-états ET). Dans cet espace, chaque état structuré, $y \in X$ et $\mathcal{T}(y) \in \{or, and\}$, définit lui-même un sous-espace d'états dénoté ST^y, une arborescence d'états enracinée en y. On peut donc dire que le STS G est lui-même *composé*

d'un ensemble de *sous-modules* dénotés G^y, des *sous-STS* dont l'espace des états est ST^y (voir [39] pour une définition formelle). La *dynamique* de ces *modules* est définie par un ensemble de holons *assignés* à chacun de leurs super-états OU, un et un seul holon par super-état OU, les états simples n'ayant aucune dynamique et la dynamique des super-états ET étant complètement déterminée par l'ensemble des super-états OU qu'ils agrègent (directement ou indirectement) en tant que *composantes ET*. Le holon constitue donc aussi une unité de modularisation en contrôle de STS, une modularisation des *comportements*. Il faut identifier lesquelles de ces unités de modularisation peuvent supporter une notion de composante réutilisable et comment.

Pour reprendre le scénario hypothétique de la section 3.1.2.1, on peut imaginer par exemple que le modèle du vérin donné en figure 3.1 (c) soit utilisé pour former une description d'*interface* similaire à celle de la figure 3.2 (a). Cette description d'interface est alors *générique* et peut être instanciée au sein d'un modèle pour une *grue* comme en figure 3.3. On peut bien sûr imaginer des mécanismes d'instanciation plus flexibles permettant de paramétrer les étiquettes d'états, d'événements ainsi que le nom du STS pour réduire le nombre d'interfaces d'une composante réutilisable, mais ce ne sont là que des considérations cosmétiques au problème.

En poursuivant l'analogie de la programmation par objets introduite en section 3.1.2.1, le modèle du système **Grue** définit donc deux composantes **Flèche1** et **Pince1**, des variables, dont le type correspond respectivement aux *classes* **Flèche** et **Pince** des stéréotypes de STS *génériques*. Au moment de l'exécution du système **Grue** chacune de ces composantes est associée à un dispositif (ou sous-système) physique correspondant à la *classe*, d'où les notions d'instance et d'instanciation. Les composantes sont considérées en fonctionnement parallèle concurrent d'où l'usage d'un super-état ET pour modéliser le système **Grue**. Les composantes sont considérées disjointes (elles ne partagent pas d'événement). On peut imaginer l'usage de noms qualifiés, pour obtenir cet effet. On a alors que (au niveau global) la composante **Grue.Flèche1** génère l'événement **Grue.Flèche1.**x**.↓X0** . La technique, bien que lourde, procure l'effet désiré. Ce schéma n'est que conceptuel, le principe est de distinguer les éléments internes de plusieurs composantes entre eux.

On peut ensuite procéder à une spécification (figure 3.4) et opérer une synthèse pour obtenir le comportement contrôlé illustré en figure 3.5.

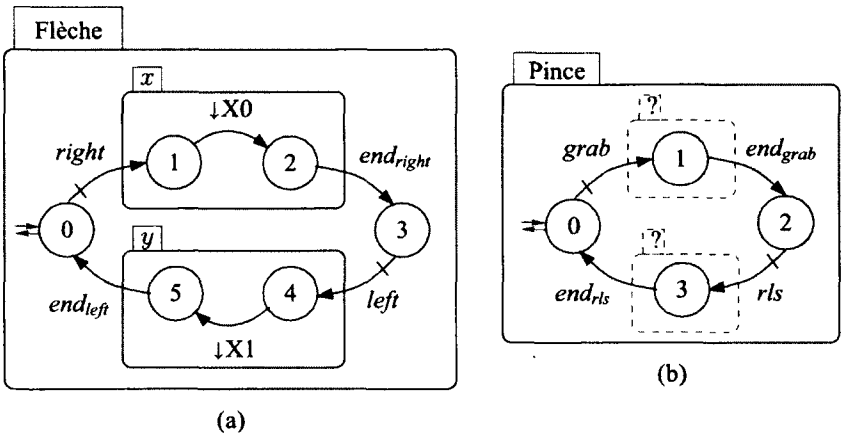


FIGURE 3.2 – STS *génériques* pour la confection d'une grue

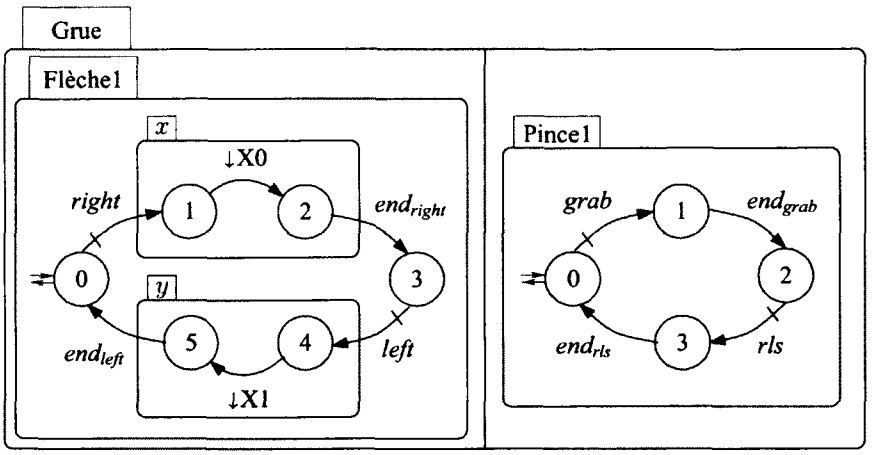


FIGURE 3.3 – STS du sous-système *Grue* (comportement libre)

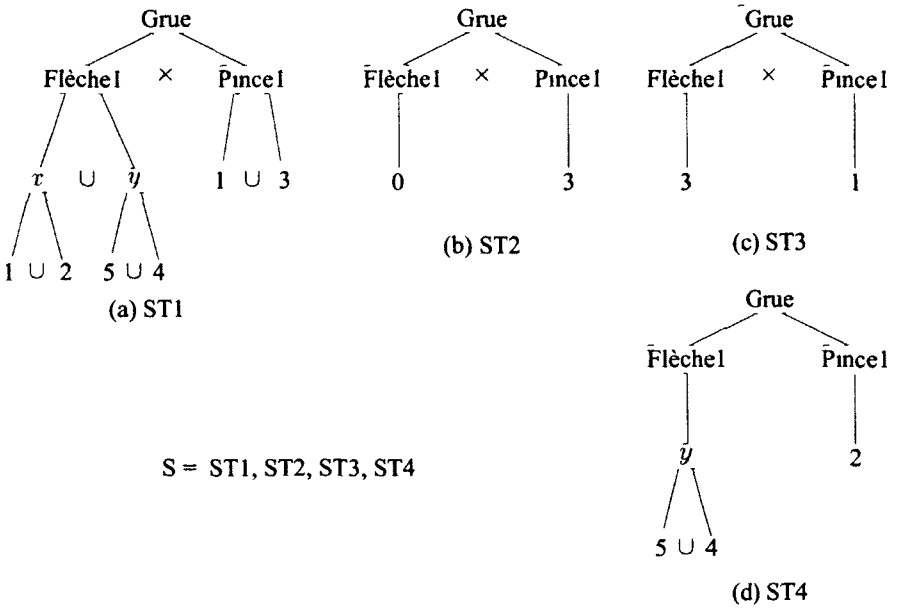


FIGURE 3.4 – Spécification du sous-système *Grue*

La spécification, illustrée en figure 3.4, définit un ensemble d'états prohibés ; des états que la grue doit éviter au cours de son fonctionnement. Elle est composée des quatre contraintes suivantes :

(a) La flèche et la pince ne peuvent pas être en fonctionnement simultanément.

(b) La grue ne peut pas déposer de pièce à la position d'entrée (flèche à gauche, état 0).

(c) La grue ne peut pas saisir de pièce à la position de sortie (flèche à droite, état 3).

(d) Une fois saisie une pièce à la position d'entrée, la grue doit nécessairement aller la déposer à la position de sortie.

La grue ne fonctionne donc que dans un sens, elle transporte des pièces de la position d'entrée à la position de sortie. Cependant elle peut osciller, à vide, entre les deux positions.

La figure 3.5 développe le *produit* des deux composantes en mode contrôlé. Plusieurs détails ont été escamotés pour ne pas alourdir la présentation. Entre autres on n'y fait pas usage de noms qualifiés, estimant le contexte suffisamment clair.

Dans la figure 3.5 on voit que certains holons, par exemple r_1 et r_2, ont été créés de toutes pièces. Comme on peut le voir dans la figure 3.6 les holons x_1 et x_2 de la grue ne sont

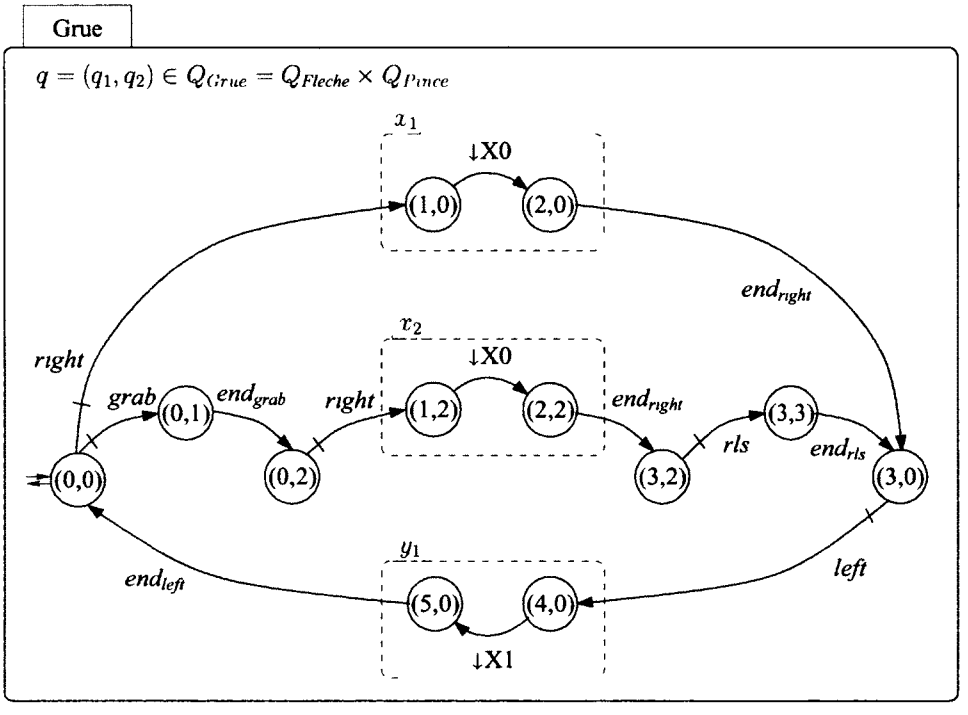


FIGURE 3.5 – Sous-système *Grue* (comportement contrôlé)

pas identiques au holon **Flèche.**x. Mais effectivement l'*interface holonique* de **Flèche.**x se *superpose* à la structure de transition du holon de la grue pour donner naissance à deux *instances* de **Flèche.**x : x_1 et x_2. La création de ces deux derniers holons (le long de la structure du holon **Flèche.**x) est nécessaire parce que x_1 et x_2 représentent deux *instances différentes* du comportement de la flèche au sein du comportement de la grue. Le contexte de leur occurrence, c'est-à-dire les états de leurs structures externe et interne ainsi que le super-état OU auquel ils sont associés, est différent. La création de ces deux holons (*nommés* différemment de **Flèche.**x) est aussi nécessaire pour maintenir le principe de *couplage local*. En effet, puisque x_1 et x_2 ne sont pas en *relation de parallélisme* ils ne peuvent pas partager l'événement **Flèche.**↓X0, d'où la nécessité des deux événements *distincts* **Grue.**x_1.↓X0 et **Grue.**x_2.↓X0. Alors, par *instanciation* du STS générique **Flèche** de la figure 3.2 on donne aussi naissance à des schémas de comportement *génériques* de cette dernière : les holons contenus dans le STS générique **Flèche**. Ces schémas de comportement sont aussi un genre d'élément réutilisable comme le démontre le STS de la grue. Mais ces schémas sont alors relatifs à une *instance spécifique* du modèle générique **Flèche**, notamment le sous-STS **Flèche1** du comportement libre de la grue (figure 3.3), car en dehors de ce dernier les *instances* x_1 et x_2 de la grue contrôlée (figure 3.5) n'ont pas de sens.

La question soulevée par cet exemple est de savoir s'il est bien raisonnable de définir des *schémas génériques de comportement* (des holons) dans l'*interface* d'un STS générique de composante comme celui de la **Flèche** en figure 3.2. Le holon *fondamental* du STS, c'est-à-dire l'AFD du modèle exhaustif de la figure 3.1 dont la structure externe est vide dans le STS **Flèche**, doit y être défini car il exprime l'essence même du comportement de la flèche : le comportement d'un vérin à commande monostable dont le fonctionnement est cyclique. Mais les holons subordonnés x et y doivent-ils faire partie de cette interface ? Bien sûr dans le modèle générique de la flèche, puisqu'il est si simple, on y voit peu d'inconvénient. On peut d'ailleurs imaginer trois définitions d'interface évidentes pour ce dispositif :

1. un STS ne contenant que le holon en forme fondamentale c'est-à-dire l'AFD de la figure 3.1 ;
2. la définition donnée en figure 3.2 (a) ;
3. et une autre définition ne comportant que l'état initial avec un holon subordonné contenant les états 1, 2, 3, 4 et 5, similaire à celui de la figure 3.7, procurant un

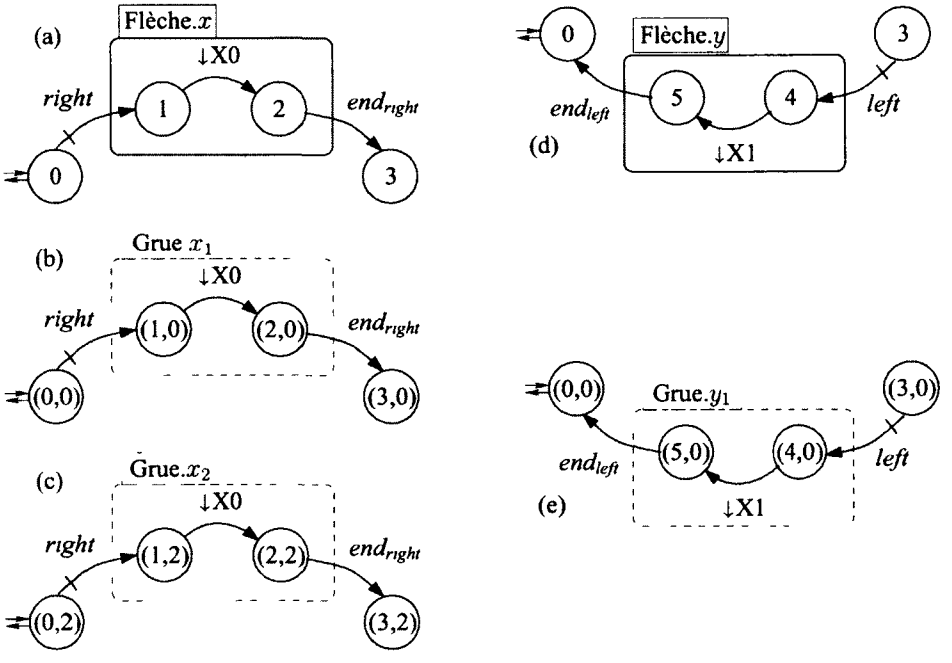


FIGURE 3.6 – Holons de la *Grue* (comportement contrôlé)

comportement du type *début de cycle-fin de cycle*.

Cet inventaire semble couvrir tout le répertoire des abstractions possibles pour une telle composante, mais il n'en est rien. Pour s'en convaincre il suffit d'examiner un *cas d'usage* fréquemment utilisé dans la conception de modèles en contrôle de STS et illustré partiellement en figure 3.7.

Dans la figure 3.7 on retrouve en (a) une composante obtenue par instanciation d'une de ses *interfaces*, comme pour la grue précédemment. La composante est utilisée au sein d'un modèle de STS constitué d'un seul super-état OU (un processus séquentiel) se superposant à un holon dont certains des états utilisent un schéma de parallélisme, illustré en (d), donc des super-états ET. Ce type de cas d'*usage* est une des caractéristiques qui fait la puissance des modèles de STS, il exprime de façon très compacte l'alternance entre parallélisme et exécution séquentielle au sein d'un procédé. Or avec l'usage de composantes réutilisables, par instanciation de STS génériques, le comportement illustré en (d) doit d'abord être spécifié comme en (b) et le résultat de la spécification élaboré comme en (c). On peut ensuite *réutiliser* ce comportement pour composer le cas d'usage (d). Or (b), (c) et (d) sont tous relatifs à la composante (a), elle-même une instance d'un STS générique (une composante réutilisable). En plus, la spécification de (c) ne peut pas être considérée comme un *sous-système* fermé utilisant (a), car (a) peut être réutilisé pour former d'autres *cas d'usage* au sein du modèle de STS en cours d'élaboration (le *système*). En effet (d) n'est qu'*une* des formes possibles pour le comportement de (a) dans ce système. Des cas d'utilisation similaires à ceux de la figure 3.6 peuvent être aussi utilisés dans ce système et le cas (d) peut y exister en plusieurs exemplaires.

Il devient évident qu'il n'est pas réaliste de requérir qu'un STS générique (une composante réutilisable) soit assorti de toutes les définitions d'*interfaces* (des holons) correspondant à tous les *cas d'usage* dans lesquels la composante peut être éventuellement impliquée. Ces cas sont trop nombreux, et dépendent généralement du problème de contrôle à résoudre. On peut avantageusement penser définir des *modèles* de holons comme celui de la figure 3.7 (d) *au sein de la solution* d'un problème de contrôle donné ou en cours d'élaboration. De tels *modèles de comportement* peuvent alors être réutilisés en tant que *schémas de comportement* lors de la solution d'un problème de contrôle. Mais leurs caractères spécifiques les disqualifient comme éléments génériques réutilisables dans le contexte

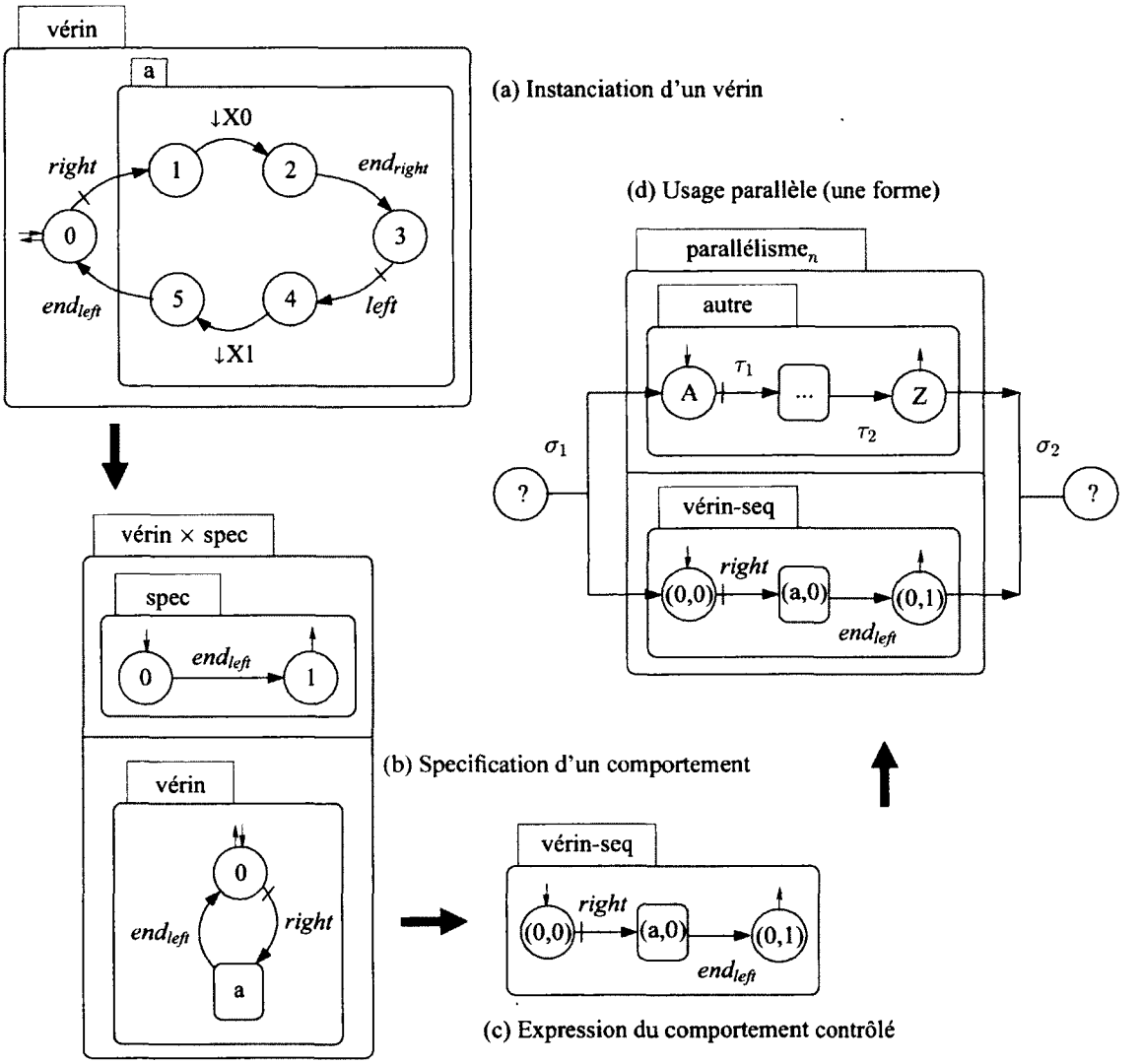


FIGURE 3.7 – Instance de comportement parallèle

général.

Le concept de holon ne semble donc pas être un élément de modularité adapté à l'expression de composantes réutilisables en contrôle de STS. La versatilité des holons et la forme compacte des modèles de STS les rendent toujours très attrayants comme notation pour l'expression de problèmes de contrôle. Mais le mode d'usage des holons, en tant que spécification d'instances de comportement d'un dispositif au sein d'un système, les rend plus similaires et mieux adaptés à la réalisation d'un outil de conception (par exemple au sein d'une interface graphique) qu'à la spécification de composantes génériques réutilisables.

Ceci semble limiter la métaphore possible pour des composantes réutilisables en contrôle de STS à des *STS génériques* en *mode fondamental*. On peut définir la notion de *STS en mode fondamental* comme un STS dont l'ensemble des holons $\mathcal{H}$ ne contient qu'un et un seul élément, dont la structure externe est vide, qui est superposé à la racine du STS x_0, un super-état OU. Un STS en mode fondamental est alors directement associé au *modèle exhaustif* d'une composante (exemple en figure 3.1) dont l'AFD constitue le holon. Un STS en mode fondamental peut être considéré comme représentant le modèle générique d'une composante réutilisable. Une telle composante réutilisable ne peut être instanciée que dans un *système* formé d'un seul super-état ET à la racine dont les *cellules* (ses composantes ET) sont alors des instances de STS en mode fondamental. Un système défini de cette façon correspond à ce que l'on appelle un *modèle plat* (composition synchrones de composantes pour définir le comportement libre) comme dans le contexte usuel de SCT.

Une question légitime à ce point est de se demander si l'on ne perd pas plus que l'on ne gagne, après tout vaut-il la peine de *sacrifier* la puissance d'expression des modèles de STS pour n'y gagner qu'une vague notion de composante réutilisable ? On trouve une réponse plus complète à cette question dans les sections et chapitres subséquents, mais on peut déjà faire remarquer ce qui suit.

Premièrement, il n'est pas ici question de *sacrifier* quoi que ce soit. Le modèle de système plat obtenu par instanciation de STS en mode fondamental (des composantes disjointes) n'est que l'expression du comportement libre du système. On doit encore y insérer les requis de contrôle qui doivent alors être représentés par des modèles de la mémoire agissant comme des contraintes ou spécifications du contrôle. Rien n'empêche durant ce

processus de raffinement (ou transcription) des requis de contrôle de procéder à l'élaboration d'un sous-système comme celui de la grue (figure 3.5). C'est là une façon de *localiser* des requis de contrôle. On peut observer que le comportement du système en boucle fermée de la grue peut lui-même constituer un STS en mode fondamental et donc une composante réutilisable.

Deuxièmement, la grue est un exemple très simple mais rien empêche d'utiliser aussi d'autres schémas de comportement plus complexe (des définitions de holons similaires à ceux de la figure 3.7) pour procéder au raffinement des requis. La seule différence est qu'il faut procéder *à partir* de l'expression du *système plat*, c'est-à-dire à partir du modèle de ses composantes car ces dernières sont les seules décrivant précisément le fonctionnement des dispositifs élémentaires impliqués. Comme, par exemple, le fait qu'un vérin pneumatique à commande monostable est un dispositif *cyclique* dont le répertoire d'actions est de *déployer* et *rétracter* le vérin. Tout autre comportement requiert l'adjonction d'une *mémoire*, c'est-à-dire une spécification, avec les complications afférentes. Or il n'est pas possible de contourner ce problème car le but de la spécification des requis de contrôle est précisément de *rendre explicite* la façon dont ces requis sont *réalisés* sur l'ensemble des dispositifs impliqués. Autrement on en est réduit à la spécification de modèles formels de haut niveau auxquels il faut *adapter* le matériel par programmation et ce, pour chaque problème de contrôle résolu de cette façon. C'est l'équivalent de l'élaboration d'un modèle *ad hoc* de la machine à contrôler adapté manuellement à la solution d'un problème de contrôle spécifique. On y perd généralement la plupart des avantages d'une méthode formelle comme SCT dans la couche insérée entre la solution de synthèse et le matériel.

L'approche préconisée dans cette étude est de réutiliser les spécifications de contrôle autant que possible par réutilisation de composantes (ou des interfaces de ces dernières). On vise à *descendre* l'approche formelle le plus près possible des dispositifs physiques pour conserver les avantages que l'approche formelle procure. Les composantes envisagées représentent éventuellement des éléments physiques à contrôler et non pas des spécifications de comportement réutilisables comme les holons de la figure 3.7. Il doit être clair que l'analyse qui précède ne suggère en rien qu'il faille abandonner quoi que ce soit dans la puissance et la versatilité de la notation et de la théorie du contrôle de STS pour y adjoindre des composantes réutilisables.

Troisièmement, on peut mentionner que s'il est possible d'élaborer des spécifications d'interfaces *opaques* de composantes réutilisables sous forme d'AFD par exemple, le modèle de STS en forme fondamental en tant que composantes réutilisables pourrait procurer un moyen de profiter des autres avantages de la théorie du contrôle de STS. On pourrait notamment profiter d'une approche par prédicats, d'une synthèse symbolique et d'un résultat de synthèse sous forme d'une SFBC tout en utilisant des composantes réutilisables *opaques* en provenance d'autres variantes de SCT mais compatibles avec un usage dans un modèle de STS. Cette avenue n'est pas non plus à négliger.

3.2.3 Sommaire d'analyse

Par rapport aux critères fixés en section 3.1, on peut dire que le contrôle de STS s'accommode bien de toutes les limitations et qu'il correspond directement aux critères C2, C6 et C7.

- Il utilise une forme d'encapsulation ouverte procurant, par le biais de ses procédures de synthèse, une consistance entre les modèles abstrait et concret (critère C2).
- C'est une approche par prédicats faisant usage de méthodes symboliques en synthèse (critère C6).
- La synthèse produit une loi de contrôle de type SFBC (critère C7).

Un STS en mode fondamental procure un élément de modularisation qu'on peut assimiler à un modèle générique instanciable et donc à une composante réutilisable (critère C1). Ce genre de composante réutilisable est cependant limité car il ne permet a priori que la composition de *systèmes plats* similaires à ceux utilisés pour l'expression du comportement libre dans le contexte usuel de SCT. Il n'empêche cependant pas la composition d'un modèle de STS plus hiérarchisé au fur et à mesure que l'on y introduit les requis de contrôle. La tâche peut cependant se révéler ardue puisque toute instance des comportements doit alors être dérivée à partir des modèles des composantes utilisés (les instances de STS en mode fondamental).

La forme des interfaces de ces STS génériques (en mode fondamental) exclut l'usage de holons subordonnés dans l'interface, excluant du même coup la nécessité d'un support formel à la dérivation d'interfaces (critère C3). Mais on peut dire que l'abstraction peut,

jusqu'à un certain point, être répétée sur l'axe vertical (critère C4) puisque l'exemple de la grue (figure 3.5) prouve qu'à partir de la composition d'instances de STS en mode fondamental, on peut obtenir d'autres modèles de STS en mode fondamental, donc d'autres composantes réutilisables.

Les procédures de synthèse sont déjà parfaitement adéquates pour des systèmes composés d'instances de STS en mode fondamental en fonctionnement en parallèle concurrent (critère C5). L'adjonction de spécifications utilisant le modèle préconisé dans la théorie (modélisation explicite de la mémoire) ainsi qu'une structuration hiérarchique éventuellement plus poussée du modèle initial (*plat*) ne présentent pas de problèmes supplémentaires en synthèse tant que les règles de modélisation des STS sont respectées.

Quant au critère C8 (support à l'implémentation), la modélisation explicite de la mémoire pour la spécification des requis de contrôle présente un avantage marqué pour l'implémentation d'une SFBC le long des lignes exposées en section 3.1.3. La structuration de l'espace des états dans les modèles de STS repose sur un principe d'agrégation hiérarchique. En conséquence chaque état du système hiérarchisé correspondant à un élément de ST correspond à un élément d'une partition de l'espace des états du modèle du *système plat*. L'ensemble des prédicats résultant de la synthèse exprime des fonctions caractéristiques sur cet espace des états du *système plat*. Cette structuration de l'espace des états et de la structure des prédicats présente aussi un avantage pour l'implémentation d'une SFBC le long des lignes exposées en section 3.1.3. La seule ombre au tableau est que le résultat de synthèse est un ensemble de prédicats sous forme de BDD dont on doit extraire éventuellement une forme utilisable dans un PLC ce qui peut représenter une difficulté substantielle.

3.3 Contrôle hiérarchique par interface

Le contrôle hiérarchique par interface (HISC, discuté en section 2.4) procure de façon directe et explicite, dans sa théorie, plusieurs des éléments discutés en section 3.1.1. Il procure déjà une division architecturale d'un problème de contrôle en *système de haut niveau* et *systèmes de bas niveau* (voir [36]). Dans cette architecture on parle souvent des systèmes de bas niveau comme de *composantes*, il faut entendre ici que ce sont des *sous-systèmes* composant le système global. Reste encore à y associer une notion de *composantes réutili-*

sables pour l'expression de problèmes de contrôle.

3.3.1 Les avantages de HISC

L'élément fondamental de modularisation dans HISC est l'*interface*. L'interface HISC *sépare* les systèmes entre eux. Le système de haut niveau ne partage, avec ses *composantes*, que la dynamique d'interface et donc *seulement* les événements des interfaces sont partagés entre le système de haut niveau et l'ensemble des systèmes de bas niveau. Outre l'ensemble des interfaces, le système de haut niveau et l'ensemble des systèmes de bas niveau sont donc, techniquement, *disjoints*. D'autre part, les systèmes de bas niveau sont tous considérés disjoints deux à deux, ils ne partagent entre eux aucun de leurs événements, même pas les événements de leurs interfaces respectives. Chaque système de bas niveau est muni d'une interface (disjointe de celle des autres systèmes de bas niveau) avec le système de haut niveau. Il en résulte une architecture *client-serveurs* où seul le système de haut niveau agit comme client et tous les systèmes de bas niveaux agissent comme serveurs. La médiation est réalisée par l'ensemble des *interfaces* et prend la forme d'un protocole *requête-réponse* auquel tous les éléments du système (client autant que serveurs) doivent se soumettre. Ces interfaces sont donc qualifiées d'interfaces en paires requête-réponse (de l'anglais *command-pair interfaces*) dont la définition est formalisée dans la théorie. Pour assurer la consistance entre client et serveurs, la théorie développe une notion de *consistance à l'interface* prenant la forme de six propriétés (ou contraintes) relatives à la forme des systèmes ainsi qu'à l'usage du protocole tant par le client que par l'ensemble des serveurs. La théorie procède ensuite à développer des notions appropriées de systèmes contrôlables et non bloquants *localement* (de l'anglais *level-wise controllable* et *level-wise nonblocking*).

Dans ce qui précède on peut distinguer assez clairement qu'une paire constituée d'un système de bas niveau muni d'une définition d'interface adéquate (c'est-à-dire respectant les propriétés pertinentes de consistance à l'interface) constitue un *module* dans un problème de contrôle. L'interface HISC procure ainsi une encapsulation *opaque* du système de bas niveau qu'elle *représente* dans le système global. Les procédures de synthèse fournies dans [9] et [52] montrent clairement qu'une synthèse dans le haut niveau est *indépendante*

d'une synthèse dans le bas niveau. Une synthèse dans le système de haut niveau ne fait intervenir que le modèle du comportement libre du système de haut niveau, l'ensemble des modèles des interfaces et l'objectif de contrôle ou la *spécification de contrôle* du système de haut niveau (tous des AFD). Une synthèse dans un des systèmes de bas niveau ne fait intervenir qu'un modèle du comportement libre du système de bas niveau, la définition de son interface et éventuellement sa spécification de contrôle.

L'innovation dans les procédures de synthèse, par rapport aux procédures correspondantes de SCT, consiste à incorporer la notion de *consistance à l'interface* comme contrainte supplémentaire dans les calculs de synthèse. Cette innovation n'a pas de conséquence majeure quant à la complexité des calculs (temps et espace polynomial). Cependant, en conséquence d'une encapsulation opaque, on y perd bien sûr l'*optimalité* (au sens théorique donné par SCT) de la solution. À noter cependant qu'on y gagne au niveau de la complexité globale de la synthèse car, en admettant effectivement que *tous* les systèmes font l'objet d'une spécification de contrôle lors de l'élaboration d'un problème de contrôle, la synthèse se décompose alors en $n + 1$ synthèses. L'ensemble de ces synthèses s'avère moins complexe qu'une synthèse globale dû à la réduction de l'espace des états qui croît de façon exponentielle avec le nombre de modèles (AFD) impliqués dans une synthèse.

Dans le cas d'une conception de problème de contrôle par composantes réutilisables, le gain potentiel est encore plus grand en ce sens que l'effort est *étalé* dans le temps. Les composantes réutilisées sont déjà munies d'interfaces et elles ont déjà, le cas échéant, fait l'objet d'une synthèse. Alors il ne reste plus qu'à *instancier* les sous-systèmes (les serveurs) à partir de leur modèles *génériques*, spécifier les requis de contrôle du système ainsi composé (le système de haut niveau) et opérer une synthèse (c'est-à-dire une synthèse dans le système de haut niveau seulement). Dans le cas d'une conception par composantes réutilisables, la réutilisation d'un modèle générique pour *instancier* un sous-système de bas niveau sous-entend la réutilisation de sa spécification de contrôle.

Finalement, des procédures de synthèse utilisant des BDD (ou compatibles avec leur usage) ont été élaborées (voir [36]). Ces procédures procurent une solution de synthèse sous forme d'une SFBC sur l'espace des états d'un *superviseur adéquat*. Song [52] élabore aussi un mécanisme crédible d'implémentation d'une telle solution.

3.3.2 Composantes réutilisables et HISC

Pour qu'il soit possible d'élaborer une métaphore de composantes réutilisables avec HISC, il faut s'assurer qu'une fois qu'un ensemble de modèles génériques sont instanciés dans un problème de contrôle, le résultat présente toujours les propriétés requises pour assurer la validité des résultats de synthèse. D'abord il faut digresser un peu pour se donner une idée plus précise de ce qu'une composante réutilisable signifie au sein de HISC.

Dans le contexte donné en section 3.1.2.1, un système est composé de plusieurs dispositifs disjoints en fonctionnement parallèle concurrent. Ainsi, sans *composante*, la notion de *système* n'a pas de sens précis. Aussi l'*instanciation* de sous-systèmes à partir de modèles génériques prend le sens d'une *déclaration de variable* d'un certain type (celui du modèle générique). Ces sous-systèmes étant *disjoints*, puisqu'on peut *instancier* plusieurs fois le même modèle générique (pour différents sous-systèmes de même *type*), chacune des *variables* déclarées définit son propre *espace symbolique* assurant que les différentes instances représentant ces sous-systèmes sont disjointes.

Dans le modèle de HISC, un sous-système (un module) est un système de bas niveau. Il est formé principalement de trois éléments, trois générateurs (des AFD) :

- un modèle de son comportement libre, la *machine* $\mathbf{G}_L^p$;
- une expression de sa loi de contrôle, le superviseur $\mathbf{S}_L$;
- et son interface $\mathbf{G}_I$.

Le comportement en boucle fermée du sous-système sous le contrôle de $\mathbf{S}_L$ (son superviseur) s'exprime par $\mathbf{G}_L = \mathbf{G}_L^p \parallel \mathbf{S}_L$. On a ainsi la paire $(\mathbf{G}_L, \mathbf{G}_I)$ qui définit, respectivement, le comportement du sous-système et son *abstraction* face au système de haut niveau (le *système* proprement dit). Puisque toutes les *instances* de la paire $(\mathbf{G}_L, \mathbf{G}_I)$ sont similaires, on peut considérer la paire comme définissant un modèle générique de composante réutilisable (une *classe*). L'*instanciation* de $(\mathbf{G}_L, \mathbf{G}_I)$, sous forme d'une déclaration de variable, redéfinit tous ses symboles dans un espace symbolique distinct des autres instances, les instances sont ainsi disjointes quant à leurs alphabets d'événements (et à leur structure de transition en général). Si on dénote les sous-systèmes en les indexant, par exemple $(\mathbf{G}_{L_j}, \mathbf{G}_{I_j})$ pour $j \in \{1. \ldots, n\}$, chaque sous-système devient une instanciation d'un modèle générique $(\mathbf{G}_L, \mathbf{G}_I)$. Pour chaque sous-système, peu importe sa *classe*, $(\mathbf{G}_{L_j}, \mathbf{G}_{I_j})$

est disjoint de $(\mathbf{G}_{L_k}, \mathbf{G}_{I_k})$ pour $j, k \in \{1, ..., n\}$ pourvu que $j \neq k$.

Ce qui précède correspond jusqu'ici au contexte défini par HISC en ce sens que tous les sous-systèmes de bas niveau possèdent des structures de transition disjointes deux à deux, autant en ce qui concerne la définition de leur interface HISC $\mathbf{G}_{I_j}$ que pour la définition de leur dynamique interne $\mathbf{G}_{L_j}$. Mais dans le cas d'une définition générique de composante réutilisable, $(\mathbf{G}_L, \mathbf{G}_I)$, comment peut-on dire qu'elle est *consistante à l'interface* ou contrôlable et non bloquante *localement*, puisque l'on n'a alors aucun système de haut niveau ? Pour rendre cette définition acceptable, il faut la considérer sous l'angle de l'*instanciation*.

On peut considérer qu'une définition de composante réutilisable (c'est-à-dire la composante générique $(\mathbf{G}_L, \mathbf{G}_I)$) suppose implicitement la définition, *au moment de son l'instanciation* en $(\mathbf{G}_{L_j}, \mathbf{G}_{I_j})$, de deux *modules* du système de haut niveau correspondant à cette *instance* :

- la définition d'un *module* du comportement libre du système de haut niveau $\mathbf{G}^p_{H_j}$;
- la définition d'un *superviseur modulaire* pour $\mathbf{G}^p_{H_j}$, le *module* $\mathbf{S}_{H_j}$.

$\mathbf{G}^p_{H_j}$ et $\mathbf{S}_{H_j}$ sont deux générateurs définis sur l'alphabet de l'interface spécifique à l'*instance* de la composante réutilisable, $\Sigma_{I_j} := \Sigma_{R_j} \dot{\cup} \Sigma_{A_j}$, et tels que $L_m(\mathbf{G}^p_{H_j}) = L_m(\mathbf{S}_{H_j}) = \Sigma^*_{I_j}$. Intuitivement on peut considérer que ces générateurs n'ont pas de structure, à savoir qu'ils sont formés d'un seul état (initial et terminal) où toute transition sur $\sigma \in \Sigma_{I_j}$ boucle simplement sur cet état. Le comportement sous contrôle du *module* correspondant du système de haut niveau est alors $\mathbf{G}_{H_j} := \mathbf{G}^p_{H_j} \parallel \mathbf{S}_{H_j}$ et il est évident que $L_m(\mathbf{G}_{H_j}) = \Sigma^*_{I_j}$ aussi. Aucun de ces *modules* n'exerce de contrainte sur le système de haut niveau. Alors seule l'interface HISC de l'*instance* de la composante réutilisable intervient comme contrainte dans le système de haut niveau.

Avec ce qui précède, on peut définir de façon plus précise ce que signifie une composante réutilisable $(\mathbf{G}_L, \mathbf{G}_I)$ (une *classe*). Il s'agit d'un modèle générique conçu de sorte qu'un système composé d'une seule et unique *instance* de $(\mathbf{G}_L, \mathbf{G}_I)$ soit *consistant à l'interface* et contrôlable et non bloquant *localement*. Ce cas de figure correspond au cas de *système séquentiel* dans [35], où $\mathbf{G}^p_H := \mathbf{G}^p_{H_j}$, $\mathbf{S}_H := \mathbf{S}_{H_j}$, $\mathbf{G}^p_L := \mathbf{G}^p_{L_j}$, $\mathbf{S}_L := \mathbf{S}_{L_j}$ et $\Sigma_I := \Sigma_{I_j}$.

On observe que ce système de haut niveau est *vide* en ce sens qu'il n'a pas de *dynamique*

propre, $\mathbf{G}_H := \mathbf{G}_H^p \parallel \mathbf{S}_H$ et $L_m(\mathbf{G}_H) = \Sigma_I^*$, c'est-à-dire que $\Sigma_H = \emptyset$. Par construction ce système respecte les propriétés (1) et (2) de la consistance à l'interface (cas séquentiel dans [35], *définition* 6). Aussi, puisque $\Sigma_H = \emptyset$, on a que $P_{IH}^{-1}(L(\mathbf{G}_H)) = P_{IH}^{-1}(\Sigma_I^*) = \Sigma^*$ et la propriété (3) est trivialement vérifiée. Le système de haut niveau *induit* par instanciation de $(\mathbf{G}_L, \mathbf{G}_I)$ (cas séquentiel) est ainsi consistant à l'interface. Il est également aisé de vérifier que le système de haut niveau, *induit* comme précédemment, satisfait la propriété (I) de la définition 4 dans [35] puisque $\mathcal{H}_m = \mathcal{H} = \Sigma^*$ et que $L(\mathbf{G}_I) = \overline{L_m(\mathbf{G}_I)}$, car $\mathbf{G}_I$ est une interface en paires requête-réponse (définition 3 dans [35]). Le système de haut niveau est donc *non bloquant* localement. Les propriétés (I) et (III) de la définition 5 dans [35] sont aussi trivialement vérifiées, (I) par construction du système *séquentiel*, et (III) par le fait que $\mathcal{H}^p = \mathcal{S}_H = \Sigma^*$. Le système de haut niveau est donc *contrôlable* localement.

La *conception* d'une composante réutilisable $(\mathbf{G}_L, \mathbf{G}_I)$ consiste alors à assurer les propriétés (4), (5) et (6) de la définition 6, la propriété (II) de la définition 4 ainsi que la propriété (II) de la définition 5 (dans [35]). Toutes ces propriétés sont relatives au sous-système de bas niveau représenté par $(\mathbf{G}_{L_j}, \mathbf{G}_{I_j})$, une *instance* unique de $(\mathbf{G}_L, \mathbf{G}_I)$, puisque la *conception* doit être réalisée dans le contexte d'un *système séquentiel* composé d'une et une seule instance de la paire $(\mathbf{G}_L, \mathbf{G}_I)$ (donc le système de haut niveau est vide, $\Sigma_H = \emptyset$). Les procédures de synthèse de [36] peuvent être utilisées à cet effet. Bien sûr, l'interface définie doit être une interface en paires requête-réponse.

À plusieurs égards ce résultat peut sembler contre-intuitif, après tout, comment un système de haut niveau peut-il être *consistant à l'interface* s'il n'en respecte pas la discipline *a priori* ($L(\mathbf{G}_H) = \Sigma_I^*$). Il faut comprendre que la seule exigence de la consistance à l'interface sur le système de haut niveau est que ce dernier doit toujours être en mesure d'accepter une *réponse* lorsque cette dernière est *éligible à l'interface*. Or dans un système de haut niveau *sans dynamique propre* ($\Sigma_H = \emptyset$) comme celui décrit précédemment, *tous les événements de réponse* sont *toujours éligibles* quelle que soit $s \in L(\mathbf{G}_H)$, puisque $L(\mathbf{G}_H) = \Sigma_I^*$. Pour mieux comprendre il est utile de considérer la forme *plane* du système global (*flat system* dans [35]). On a que **Plant** $:= \mathbf{G}_H^p \parallel \mathbf{G}_L^p = \mathbf{G}_L^p$ puisque $\mathbf{G}_H^p$ est un AFD acceptant Σ_I^* et que $\mathbf{G}_L^p$ est défini sur $\Sigma_{IL} = \Sigma_I \dot{\cup} \Sigma_L$. On a aussi le superviseur *plat* **Sup** $:= \mathbf{S}_H \parallel \mathbf{S}_L \parallel \mathbf{G}_I = \mathbf{S}_L \parallel \mathbf{G}_I$ puisque $\mathbf{S}_H$ est un AFD acceptant Σ_I^* et que le produit $\mathbf{S}_L \parallel \mathbf{G}_I$ est aussi défini sur Σ_{IL}. Autrement dit, le comportement du système global appa-

raît alors comme le comportement contrôlé du sous-système qu'il contient. Pour le système de haut niveau, ce comportement apparaît comme le *comportement libre* de l'interface de son sous-système, $L_m(\mathbf{G}_I)$, un comportement *localement contrôlable* et *localement non bloquant* (localement au système de haut niveau).

Qu'arrive-t-il maintenant si on compose n *instances* de ce genre de composantes ? On a un ensemble d'instances $(\mathbf{G}_{L_j}, \mathbf{G}_{I_j})$ $(j \in \{1, \ldots, n\})$ obtenues de descriptions génériques habituellement diverses mais où deux instances (ou plus) peuvent provenir de la même description générique.

Chaque instance de composante réutilisable $(\mathbf{G}_{L_j}, \mathbf{G}_{I_j})$ définit *implicitement* un ensemble de trois *modules* du système de haut niveau, $\mathbf{G}_{H_j} := \mathbf{G}^p_{H_j} \parallel \mathbf{S}_{H_j}$, comme mentionné précédemment. Pour une instance $(\mathbf{G}_{L_j}, \mathbf{G}_{I_j})$ ces trois *modules* sont définis sur l'alphabet Σ_{I_j} où pour $j \neq k$ $(j, k \in \{1, \ldots n\})$ Σ_{I_j} et Σ_{I_k} sont disjoints. Il en résulte pour le système de haut niveau trois générateurs

$$\begin{aligned} \mathbf{G}^p_H &:= \mathop{\Big\|}_{j \in \{1 \ldots n\}} \mathbf{G}^p_{H_j} \\ \mathbf{S}_H &:= \mathop{\Big\|}_{j \in \{1 \ldots n\}} \mathbf{S}_{H_j} \\ \mathbf{G}_H &:= \mathop{\Big\|}_{j \in \{1 \ldots n\}} \mathbf{G}_{H_j} \end{aligned}$$

tous définis sur l'alphabet

$$\Sigma_I := \bigcup_{j \in \{1 \ldots n\}} \Sigma_{I_j}$$

de l'ensemble des interfaces d'*instance*. De toute évidence un tel système de haut niveau n'a pas de *dynamique propre*, c'est-à-dire $L_m(\mathbf{G}_H) = L(\mathbf{G}_H) = L(\mathbf{G}^p_H) = L(\mathbf{S}_H) = \Sigma_I^*$, ainsi $\Sigma_{IH} = \emptyset$ et encore une fois $\mathcal{H} = P_{IH}^{-1}(L(\mathbf{G}_H)) = P_{IH}^{-1}(\Sigma_I^*) = \Sigma^*$. On a aussi que $\mathcal{H}_m = P_{IH}^{-1}(L_m(\mathbf{G}_H)) = P_{IH}^{-1}(\Sigma_I^*) = \Sigma^*$. Le système de haut niveau n'ayant aucune *dynamique propre*, il est nécessairement consistant à chacune des interfaces individuellement (voir [9] en section 3 pour une définition précise).

À ce point il est instructif de considérer maintenant la forme *plane* du système global

obtenu de ce qui précède (telle que définie dans [38]). On a que

$$\begin{aligned} \textbf{Plant} &= \mathbf{G}^p_H \parallel \mathbf{G}^p_{L_1} \parallel \ldots \parallel \mathbf{G}^p_{L_n} \\ \textbf{Sup} &= \mathbf{S}_H \parallel \mathbf{S}_{L_1} \parallel \ldots \parallel \mathbf{S}_{L_n} \parallel \mathbf{G}_{I_1} \parallel \ldots \parallel \mathbf{G}_{I_n}. \end{aligned}$$

Or $L(\mathbf{G}^p_H) = L(\mathbf{S}_H) = \Sigma_I^*$ et chacun des produits $\mathbf{G}^p_{L_1} \parallel \ldots \parallel \mathbf{G}^p_{L_n}$ et $\mathbf{S}_{L_1} \parallel \ldots \parallel \mathbf{S}_{L_n}$ donne un résultat dans Σ_{IL}^* où Σ_{IL} est donné par

$$\Sigma_{IL} := \bigcup_{j \in \{1,\ldots,n\}} (\Sigma_{I_j} \dot{\cup} \Sigma_{L_j})$$

et contient Σ_I. De plus le produit $\mathbf{G}_{I_1} \parallel \ldots \parallel \mathbf{G}_{I_n}$ est défini sur Σ_I. Ainsi les expressions du comportement libre et du superviseur se simplifient comme suit

$$\begin{aligned} \textbf{Plant} &= \mathbf{G}^p_{L_1} \parallel \ldots \parallel \mathbf{G}^p_{L_n} \\ \textbf{Sup} &= \mathbf{S}_{L_1} \parallel \ldots \parallel \mathbf{S}_{L_n} \parallel \mathbf{G}_{I_1} \parallel \ldots \parallel \mathbf{G}_{I_n}. \end{aligned}$$

Le système contrôlé prend alors la forme

$$\textbf{Plant} \parallel \textbf{Sup} = \mathop{\Vert}_{j \in \{1,\ldots,n\}} \left(\mathbf{G}^p_{L_j} \parallel \mathbf{S}_{L_j} \parallel \mathbf{G}_{I_j} \right).$$

Chacun des produits $(\mathbf{G}^p_{L_j} \parallel \mathbf{S}_{L_j} \parallel \mathbf{G}_{I_j})$ définit un générateur disjoint de tous les autres. Par conception de la composante réutilisable dont il est une instance, chaque générateur est consistant à son interface et localement contrôlable et non bloquant. Cette situation est analogue à la composition modulaire de sous-systèmes dans le contexte usuel de SCT où l'expression du comportement libre du système global se résume au produit synchrone du comportement libre de ses *composantes*.

À ce système, il ne reste plus qu'à définir un objectif de contrôle E_{II} (implicitement un sous-langage de Σ_I^*) et à lui appliquer les procédures de synthèse (pour un système de haut niveau) de [36] pour obtenir une solution.

Cette définition de composantes réutilisables pour HISC présume que le système de haut niveau n'est en fait constitué que de la dynamique des interfaces HISC des sous-

systèmes qui le composent. Autrement dit $\Sigma_H = \emptyset$ dans le contexte HISC. Cette restriction peut sembler curieuse, mais elle cadre bien avec la définition intuitive d'un système composé de dispositifs physiques considérés en fonctionnement parallèle concurrent comme c'est le cas dans les usines modulaires. En effet si on définit un système de haut niveau *non vide* à l'origine, $\Sigma_H \neq \emptyset$, il doit nécessairement être constitué d'un certain nombre d'instances de modèles génériques tels que l'on a vu précédemment. Appelons cet ensemble de sous-systèmes l'ensemble fondamental. Alors il n'y a pas de perte de généralité à poser un système de haut niveau vide et à y incorporer l'ensemble des instances de composantes réutilisables constituant l'ensemble fondamental en plus des autres sous-systèmes devant constituer le système global. Tout au plus, on doit incorporer au modèle global l'ensemble des contraintes s'appliquant à l'ensemble fondamental et procurant son comportement original en tant que système de haut niveau.

3.3.3 Sommaire d'analyse

Par rapport aux critères fixés en section 3.1, on peut dire que HISC s'accommode bien de toutes les limitations et qu'il correspond directement aux critères C1, C2, C5, C6 et C7.

- La notion de système de bas niveau doté d'une interface constitue un élément de modularisation adéquat pour articuler celle de composante réutilisable.
- L'interface procure une encapsulation opaque de la dynamique de la composante simplifiant significativement la synthèse et la perte d'optimalité conséquente est en pratique raisonnable.
- Les procédures de synthèse développées dans [36] sont adaptées à la métaphore de composantes réutilisables articulée en section 3.3.2.
- Des procédures de synthèse utilisant les méthodes symboliques (avec des BDD) ont été développées par Song [52].
- Les procédures développées par Song produisent une SFBC comme résultat de synthèse.

Il faut noter cependant que, même si des procédures de synthèse utilisant des BDD ont été développées, la spécification des requis de contrôle utilise toujours des AFD définissant un *langage légal* (cas du contexte usuel de SCT). Cette façon de procéder est particu-

lièrement difficile à utiliser efficacement et réduit la clarté de l'expression des requis de contrôle. Avec des AFD de *langage légal* la traçabilité des requis est quasi impossible. Il y aurait avantage à pouvoir introduire une façon de spécifier les requis directement en termes de familles d'états prohibés comme pour les STS.

Dans la même ligne, on peut mentionner que la mémoire n'est pas modélisée explicitement (sous forme de contraintes) dans la spécification des requis de contrôle (comme dans le contexte des STS). Bien sûr, l'expression d'un AFD de *langage légal* exprime *implicitement* une forme de mémoire lorsqu'une mémoire est requise. Le résultat de synthèse, un superviseur adéquat, *matérialise* alors cette mémoire. Mais alors, pour fins d'implémentation (critère C8), on est contraint d'implémenter la structure de transition de ce superviseur adéquat *au complet* puisque les éléments de mémoire y sont *implicites*. Dans la structure de transition d'un superviseur adéquat, on ne peut pas distinguer entre ce qui a trait au comportement des dispositifs physiques et ce qui requiert une mémoire. L'espace des états des superviseurs adéquats a tendance à être si volumineux qu'en général leur implémentation n'est pas réalisable, en particulier sur un automate programmable.

Le processus d'abstraction en composantes réutilisables ne peut pas vraiment être répété sur l'axe vertical (critère C4). HISC est pour l'instant limité à une description de problèmes de contrôle en deux couches, un système de *haut niveau* faisant usage de n systèmes de *bas niveau*. Bien qu'il soit possible éventuellement de composer quelques-uns de ces systèmes de *bas niveau* en composantes réutilisables, dans le cadre théorique actuel, cette manoeuvre implique de rédiger de nouvelles spécifications pour la composante ainsi constituée. C'est-à-dire que l'on ne réutilise pas directement l'ensemble des spécifications déjà obtenu pour ces systèmes de *bas niveau*, le cadre théorique ne le permet pas.

En ce qui concerne la dérivation d'interfaces, on dispose d'un certain support théorique (critère C3). Ce support théorique se limite cependant à une spécification de *langage légal* suivi d'une synthèse SCT pour obtenir une loi de contrôle adéquate relativement à la description d'interfaces. Les modèles pour la *machine de bas niveau* (modèles de comportement libre) sont supposés provenir directement d'un processus de modélisation *ad hoc* comme celui de la section 3.1.1.2. Or la recherche d'une interface adéquate demeure un processus largement heuristique puisqu'il s'agit de trouver une *projection naturelle* (section 1.1.2) du comportement en boucle fermée du système de bas niveau. Comme

mentionné dans [56] le calcul d'une projection naturelle pour un modèle d'AFD donné est en général de complexité exponentielle et donne souvent une structure de transition dont l'espace d'états est plus volumineux que l'AFD original. Aussi la modélisation d'interface HISC doit souvent faire appel à des heuristiques (illustré en figure 3.8).

La figure 3.8 illustre la confection d'une interface en paires requête-réponse pour une perceuse. La perceuse coordonne trois machines : un moteur électrique, un étau et un patin. L'ensemble admet une mesure de parallélisme entre le démarrage du moteur et le serrage d'une pièce à percer et aussi au desserrage et à l'arrêt du moteur, autrement le processus est séquentiel. Au niveau du système il n'est pas nécessaire de suivre toutes les étapes, seules la commande de démarrage et la notification d'arrêt sont nécessaires. Or en présence de parallélisme il est très difficile de trouver une projection naturelle ayant la forme d'une interface HISC (une ou plusieurs paires requête-réponse en séquence). La solution souvent adoptée (voir l'exemple de modélisation AIP dans [37, 34] entre autres) est de séquentialiser toutes les opérations. Une autre heuristique souvent utilisée est illustrée dans l'exemple de la perceuse. Ici elle permet d'*encapsuler* le parallélisme entre l'étau et le démarrage et l'arrêt du moteur. On y définit une machine fictive au seul but d'introduire des événements synthétiques (qui n'existent que dans le contexte du sous-système *perceuse*). La spécification de contrôle incorpore ces événements pour définir le début et la fin de la séquence de perçage et, comme ces événements sont *synthétiques*, ils apparaissent à point nommé pour que l'on puisse définir la projection naturelle correspondant à la structure de l'interface. Notez qu'il est nécessaire que l'événement d'arrêt (l'événement de réponse dans l'interface) soit contrôlable. C'est une condition nécessaire pour obtenir une projection naturelle qui correspond à l'interface de la figure 3.8.

L'usage d'artefacts de modélisation tels que les *événements synthétiques* est sûrement commode pour en arriver à une spécification d'interface. Elle constitue à la fois une façon d'étendre le répertoire d'événements d'une composante (réouverture de la composante une fois constituée) et une façon acceptable de limiter la liberté de modélisation sous forme d'heuristiques *disciplinées*.

Les sous-systèmes HISC ont cependant une caractéristique fâcheuse. Il n'est pas possible de gérer dans un sous-système un événement incontrôlable pouvant se produire de façon asynchrone n'importe quand. Un tel événement ne peut pas constituer un événement

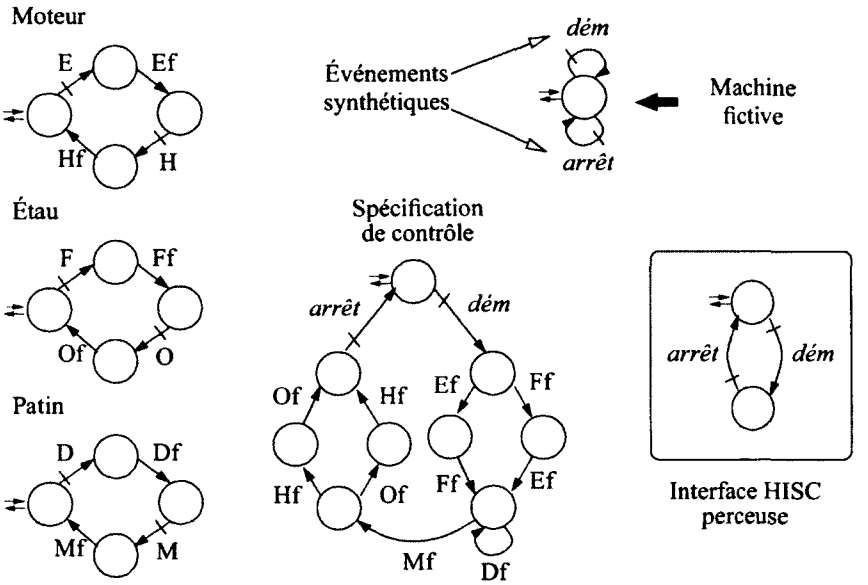


FIGURE 3.8 – Heuristiques de modélisation HISC

de *réponse* dans l'interface puisqu'il peut se produire *pendant* l'exécution d'une *requête* et qu'en général il ne termine pas la requête, la discipline d'interface est alors violée. Si le résultat de l'occurrence d'un tel événement doit être visible au haut niveau, l'heuristique des *événements synthétiques* est alors utilisée pour introduire une *interrogation* du système de bas niveau (voir l'exemple AIP précité). Le système de haut niveau interroge alors continuellement le sous-système dans un genre de boucle d'*attente active*. L'effet global cependant est de transporter un requis de contrôle aisément réglé dans le bas niveau vers le niveau supérieur. Ce genre de manoeuvre *enrichit* le niveau supérieur d'un nombre de requis de contrôle qui ont avantage à ne pas y apparaître car ils sont vraiment de portée locale.

De façon générale l'usage de projections naturelles pour les interfaces HISC a comme conséquence de rendre la structure de transition de l'interface inutilement *riche*. On peut prendre la grue de la figure 3.5 comme exemple. Cette grue n'a que trois fonctions : dégager la position d'entrée (*right*), dégager la position de sortie (*left*) et transborder une pièce (la portion centrale *grab*). Or la séquence de transbordement d'une pièce comporte aussi l'occurrence d'un événement *right*. Ainsi seul le graphe entier de la figure 3.5 constitue une

projection naturelle correspondant à une interface HISC (il faut bien sûr effacer les événements ↓X0 et ↓X1). Encore une fois, certains requis sont *poussés vers le haut*, obscurcissant l'ensemble des requis de contrôle (en augmentant leur complexité dans le système de haut niveau) alors que ces requis sont clairement de portée locale.

Finalement il est intéressant de noter qu'un système HISC constitue un système *plat*, tout au moins avec la métaphore de composantes réutilisables proposée. La capacité de réutilisation de sous-systèmes (assemblage de sous-systèmes en unités plus complexes) y est aussi limitée. On ne pourrait pas par exemple reprendre la perceuse de la figure 3.8 et la composer directement avec d'autres sous-systèmes du même genre pour définir une autre composante réutilisable (un autre modèle générique). Dans ce cas, la seule réutilisation possible consiste à ré-assembler une nouvelle composante à partir des modèles de bas niveau, ce qui constitue une limitation assez sévère. On ne peut pas construire de nouvelles abstractions à partir d'autres abstractions déjà définies. Ainsi on peut dire que seuls les *modèles exhaustifs* de la section 3.1.1.2 sont réellement réutilisables avec HISC.

3.4 Contrôle hiérarchique

Le contrôle hiérarchique (discuté en section 2.3) a été originalement étudié par Zhong et Wonham dans [66] et [65]. Ces investigations préliminaires ont été réalisées en présumant le contexte de la théorie originale de Ramadge et Wonham (voir [43, 44, 45]) et en utilisant le modèle du contrôle hiérarchique de la figure 2.1. On trouve un sommaire de ces travaux dans les notes de Wonham [62] au chapitre 5 où ils sont présentés comme une extension mineure du contexte original. L'emphase de ces travaux est axée sur une réduction de la complexité en synthèse pour procurer un moyen de mitiger le problème de l'explosion combinatoire de l'espace des états. Ces travaux se concentrent donc sur le contrôle hiérarchique en tant que *technique* de réduction de la complexité dans la solution de problèmes de contrôle n'impliquant qu'un seul système.

Plus tard Wong, dans sa thèse de doctorat [55], a donné au contrôle hiérarchique sa propre base algébrique et généralisé les notions de la théorie originale de SCT pour ce contexte. La section 2.3 reflète principalement le contexte formel établi par Wong dans sa thèse de doctorat dont on trouve une formulation plus compacte dans [58, 59] et un nombre

de séquelles intéressantes dans [60], [57] et [56].

3.4.1 Les avantages du contrôle hiérarchique

Comme mentionné précédemment le contrôle hiérarchique a souvent été présenté dans le contexte de l'abstraction et du contrôle d'un seul système. On a semble-t-il rarement tenté de l'examiner sous l'angle de la composition de plusieurs systèmes en fonctionnement parallèle concurrent disposant chacun de leur propre *abstraction*. Même les travaux de Wong à ce sujet (par exemple sur le contrôle modulaire et distribué dans [59] et [55] au chapitre 4) ne mettent pas l'emphase directement sur les possibilités qui semblent s'offrir dans un tel contexte. Or, comme mentionné par Wonham dans [62], l'abstraction hiérarchique selon le modèle de la figure 2.1 n'est pas limitée à un modèle à deux niveaux. Approximativement, dans les termes de [62] :

> Une fois établie les conditions de consistance hiérarchique nécessaires entre deux niveaux (disons G_0 et G_1), le processus peut être répété en prenant G_1 comme système de bas niveau...

Cette dernière citation est particulièrement intéressante puisqu'elle semble ouvrir la perspective à des hiérarchies *pyramidales* de sous-systèmes en fonctionnement parallèle concurrent constituant un système complet. À elle seule cette possibilité vaut certainement une investigation détaillée du sujet, particulièrement dans le cadre d'une étude sur l'utilisation de *composantes* réutilisables en contrôle supervisé.

Une architecture hiérarchique pyramidale de *sous-systèmes disjoints* en fonctionnement parallèle concurrent se prête aisément à une modularisation en composantes réutilisables. Il y a alors une forme de modularisation des requis de contrôle et surtout une réutilisation telle quelle de l'expression de ces requis ainsi que de leur implémentation. Il est aussi possible, en général, de réduire la taille des *abstractions* (modèles d'AFD des systèmes de haut niveau) par rapport à celle des *agents* (modèles d'AFD des systèmes de bas niveau). On en tire une réduction de la complexité en synthèse car le contrôle hiérarchique procure une forme d'encapsulation opaque. L'encapsulation opaque peut cependant entraîner la perte de l'*optimalité* (section 1.2.2) de la synthèse.

Une telle architecture offre en plus la possibilité d'exprimer les requis du contrôle d'un système de façon progressive et de les localiser le plus près possible de leur point d'application. Cette forme de modularisation des requis de contrôle procure une façon progressive de concevoir un système par raffinements successifs. Elle permet aux concepteurs de se concentrer sur la formulation de sous-ensembles restreints de requis de contrôle à chaque itération, divisant ainsi le problème global en sous-problèmes particuliers (une technique souvent appelée *diviser-pour-régner*).

La modularisation des requis de contrôle peut aussi procurer une clarification significative dans l'expression des requis de contrôle. Il est assez bien connu (quoique rarement souligné) que la *spécification de sous-langages légaux*, utilisée pour exprimer les requis de contrôle dans le contexte original de SCT, présente des difficultés significatives. Plus ces spécifications sont volumineuses (plus elles couvrent de requis de contrôle simultanément) moins elles sont faciles à tracer à leurs requis informels d'origine (par exemple voir [37] pour un problème type).

On remarque que le problème de synthèse aussi prend alors un caractère *progressif*. Certains requis de contrôle peuvent présenter un caractère purement *local* en ce sens qu'ils ne s'appliquent qu'à des sous-systèmes et que le contrôle du système global présuppose l'application préalable de ces requis. On peut ainsi imaginer des situations où, une fois un sous-système composé, on lui applique un ensemble de requis de contrôle de portée purement locale et on en fait la résolution par synthèse (un sous-problème de contrôle). En contrôle hiérarchique il est possible d'encapsuler les sous-systèmes (après synthèse) au même titre et de la même façon que les composantes réutilisables (par abstraction). La frontière entre composantes réutilisables et sous-systèmes devient ainsi floue. Un sous-système encapsulé de cette façon peut aisément être transformé en une composante réutilisable.

Finalement on peut noter que, dans le contexte théorique de Wong et Wonham [58], le contrôle hiérarchique peut utiliser n'importe quelle procédure de synthèse pourvu que le système de haut niveau (l'*abstraction*) dispose d'une technologie de contrôle standard (section 2.3.1.3). On peut donc y appliquer les procédures de synthèse du contexte original de SCT (Ramadge et Wonham) ou, moyennant une modélisation adéquate, celles de la théorie du contrôle de STS. Ceci peut rendre cette approche particulièrement versatile puisqu'alors rien n'empêche d'utiliser des composantes réutilisables conçues à partir de la

théorie du contrôle hiérarchique dans d'autres contextes théoriques (par exemple STS ou HISC). La réduction de complexité implicite au contrôle hiérarchique représente déjà un apport substantiel.

En somme le contrôle hiérarchique semble posséder beaucoup de caractéristiques désirables pour établir une métaphore de composantes réutilisables. Cependant, étant donné le contexte des précédentes études sur le sujet (celui de l'abstraction d'un système unique), il faut d'abord développer certains résultats théoriques pour supporter le modèle d'abstraction suggéré à la section 3.1.2.1.

3.4.2 Composantes réutilisables en contrôle hiérarchique

Dans le cadre de notre problème de conception de composantes réutilisables, le point de départ est toujours un générateur $\mathbf{G}$ avec $L = L(\mathbf{G})$, où $L \subseteq \Sigma^*$ est un langage préfixe clos muni d'une technologie de contrôle standard $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$. Cette technologie de contrôle induit $\mathcal{C}_{lo} : \mathcal{F}_L \rightarrow \mathcal{P}^2(L)$ une structure de contrôle standard localement définissable sur L (Wong [55], section 3.1). Dans le contexte hypothétique de la section 3.1.2.1, le générateur $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ représente l'AFD d'un *modèle exhaustif* obtenu par un processus *ad hoc* comme celui de la figure 3.1. En contrôle hiérarchique $\mathbf{G} = \mathbf{G}_{lo}$ et constitue ce qu'il est convenu d'appeler l'*agent* par opposition à l'*abstraction* dans le système. En général on suppose que les modèles d'AFD générateurs sont *soignés* de sorte qu'on a, pour un *agent*, le modèle linguistique (L, L_m) où $L = \overline{L_m}$ avec $L = L(\mathbf{G}_{lo})$ et $L_m = L_m(\mathbf{G}_{lo})$.

Le *mécanisme* de l'abstraction en contrôle hiérarchique prend la forme d'une projection causale $\theta : L \rightarrow M$ (section 2.3.2.1) où $M \subseteq T^*$ est défini sur un alphabet T éventuellement disjoint de Σ. On suppose en général qu'on peut obtenir pour θ une représentation sous forme d'un AFD générateur $\mathcal{G} = (Y, T, \eta, y_0, Y_m)$ où $L(\mathcal{G}) = \theta(L) = M$. On a naturellement que $M \subseteq T^*$ est fermé puisque θ est causale et L est fermé. En contrôle hiérarchique $\mathcal{G} = \mathbf{G}_{hi}$ et constitue ce qu'il est convenu d'appeler l'*abstraction* par opposition à l'*agent* dans le système. On a donc le modèle linguistique (M, M_m) pour l'*abstraction* où $M = \overline{M_m}$ avec $M = L(\mathbf{G}_{hi})$ et $M_m = L_m(\mathbf{G}_{hi})$ puisqu'on peut aussi, sans perte de généralité, supposer que $\mathbf{G}_{hi}$ est *soigné*.

Dans le scénario de la section 3.1.2.1, pour que l'on puisse considérer que $\mathbf{G}_{hi}$ consti-

tue une *abstraction* de $\mathbf{G}_{lo}$ (l'*agent*), la projection θ ne peut pas être arbitraire. Elle doit posséder plusieurs caractéristiques. En particulier l'*abstraction* doit aussi être munie d'une structure de contrôle $\mathcal{C}_{hi} : \mathcal{F}_M \rightarrow \mathcal{P}^2(M)$ et cette structure de contrôle doit être *consistante* avec celle de l'*agent*, $\mathcal{C}_{lo}$. La projection θ doit donc être *construite* de façon à ce que le triplet $(L, \theta, \mathcal{C}_{lo})$ présente la consistance du contrôle (section 2.3.2.2). Mais la consistance du contrôle, bien que nécessaire, ne suffit pas.

1. Il faut pouvoir combiner plusieurs *abstractions* pour former une description *abstraite* du problème de contrôle à résoudre. La composition de modèles abstraits doit se faire par usage du *produit synchrone* tel que défini en section 1.1.4.
2. Il faut aussi pouvoir effectuer la synthèse sur la description *abstraite* du problème de contrôle, ce qui pose le problème de la disponibilité de procédures de synthèse au niveau des représentations abstraites des composantes.
3. Il faut pouvoir effectuer une synthèse contrôlable et non bloquante sur le modèle *abstrait* du problème de contrôle et disposer de l'assurance d'une implémentation contrôlable et non bloquante au niveau *concret* (celui des *agents*).
4. Il faut enfin qu'un résultat de synthèse au niveau *abstrait* puisse être *transcrit* au niveau *concret* de façon algorithmique sans passer par une synthèse sur l'ensemble des *agents* impliqués dans la formation d'un système.

L'annexe C développe l'ensemble des résultats nécessaires pour obtenir ces caractéristiques, en utilisant la notation et les concepts présentés en section 2.3 et plus particulièrement le modèle du contrôle hiérarchique de la section 2.3.2.

3.4.2.1 Abstraction de composantes en contrôle hiérarchique

L'ensemble des résultats de l'annexe C gravite autour de trois éléments : une règle pour la *conception* de la projection (l'équation C.5) et deux propriétés dont la projection doit être dotée : la propriété d'observateur (section 2.3.3) et la *coïncidence du contrôle* (section B.3).

L'ensemble des règles définies par l'équation C.5 (section C.3)

$$\begin{aligned} X_\tau &:= \{s\sigma \in L_{voc} \mid \omega(s\sigma) = \tau\} \\ T_\theta &:= \{\tau \in T \mid X_\tau \neq \emptyset\} \\ T_c &\subseteq \{\tau \in T_\theta \mid X_\tau \subseteq \Sigma^*\Sigma_c\} \\ T_u &:= T_\theta - T_c \end{aligned}$$

assurent que la projection ($\theta : L \rightarrow M$) induit une technologie de contrôle standard ($T = T_c \dot{\cup} T_u$) dans l'*abstraction*. Sans perte de généralité on suppose ici que $T = T_\theta$. Cette condition est nécessaire puisque l'on veut pouvoir effectuer la synthèse sur le modèle abstrait du problème de contrôle et que les procédures de synthèse définies pour SCT présument toutes une technologie de contrôle standard. Alors l'abstraction (que l'on désigne souvent par son langage fermé M) est ainsi munie d'une structure de contrôle standard $\mathcal{C}_{hi} : \mathcal{F}_M \rightarrow \mathcal{P}^2(M)$.

Comme on peut le constater, dans l'ensemble des règles, la technologie de contrôle standard de $\mathcal{C}_{hi}$ est obtenue en imposant un critère simple pour qu'un événement soit considéré contrôlable dans l'*abstraction* : $T_c \subseteq \{\tau \in T_\theta \mid X_\tau \subseteq \Sigma^*\Sigma_c\}$. L'événement τ n'est donc considéré contrôlable dans l'*abstraction* que s'il est possible d'en *inhiber* la génération à travers l'*agent*, et ce, en toutes circonstances. Le reste des événements (ceux qui ne répondent pas à ce critère) sont tout simplement considérés incontrôlables dans l'abstraction : $T_u := T - T_c$. En d'autres termes on obtient l'*absence de délai de contrôle*

$$(\forall \tau \in T_c)\ \emptyset \neq \omega^{-1}(\tau) \subseteq \Sigma^*\Sigma_c \text{ (cf. formule C.6).}$$

Pour toute chaîne $s\sigma \in L$ telle que $\theta(s\sigma) = \theta(s)\tau \in M$ et $\tau \in T_c$ alors $\sigma \in \Sigma_c$. La génération de τ à la suite de la génération de $\theta(s)$ peut donc être contrôlée à travers une action de contrôle *directe* sur l'ensemble d'événements $\{\sigma \in \Sigma_c \mid \theta(s\sigma) = \theta(s)\tau\}$ dans l'*agent*.

L'usage de l'ensemble des règles de l'équation C.5 lors de la construction d'une projection $\theta : L \rightarrow M$ induit donc une technologie de contrôle standard dans l'abstraction et procure l'*absence de délai de contrôle*. La proposition C.11 montre qu'en présence de la

propriété d'observateur cette situation (l'absence de délai de contrôle) entraîne la *coïncidence du contrôle*

$$\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$$

(la relation entre ces deux concepts est discutée en section B.3). Lorsque la projection θ est ainsi dotée de la *coïncidence du contrôle* et de la propriété d'observateur :

a) le corollaire C.5, le corollaire C.2 et la proposition C.4 à la section C.1 démontrent que les conditions correspondant à la version faible de la *consistance du contrôle* sont réunies, c'est-à-dire que $\mathcal{C}_{hi}(M) \subseteq \theta(\mathcal{C}_{lo}(L))$;

b) la proposition C.8 à la section C.2 démontre que les conditions correspondant à l'*absence de couplage de contrôle* sont aussi réunies et donc $\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$.

Il suffit alors de poser $L'_m := \theta^{-1}(M_m)$ pour obtenir la *consistance du marquage* (voir l'argumentation en section C.2 et l'équation C.4). Ainsi les conditions du théorème 6 concernant la synthèse non bloquante en présence de la version faible de la consistance du contrôle (telles que développées en annexe A) sont alors réunies.

Pour un système hiérarchique (L, M, θ) où θ est une projection causale dotée de la propriété d'observateur conçue en respectant les règles de l'équation C.5, l'implémentation dans L d'un résultat de synthèse non bloquant sur M est aussi non bloquante. En particulier pour $E \subseteq M$ un objectif de contrôle, $\theta^{-1}(\kappa_M(E))$ est contrôlable et non bloquant (voir le théorème C.10). Le résultat de synthèse, une loi de contrôle sur l'*abstraction* $V_N : M \to \mathcal{P}(T)$, permet (à travers une règle simple, l'équation C.7) le *calcul* d'une loi de contrôle $V_K : L \to \mathcal{P}(\Sigma)$ sans passer par une synthèse sur l'*agent* (voir la proposition C.13 en section C.3). Dans ces circonstances, pour $\mathbf{G}_{lo}$ et $\mathbf{G}_{hi}$ définis comme précédemment, la paire $(\mathbf{G}_{lo}, \mathbf{G}_{hi})$ où $L = L(\mathbf{G}_{lo})$ et $M = \theta(L) = L(\mathbf{G}_{hi})$, constitue un modèle générique adéquat pouvant servir de base à une instanciation du système hiérarchique (L, M, θ), moyennant un mécanisme adéquat comme celui de la section 3.1.2.1. Alors $\mathbf{C} = (\mathbf{G}, \mathcal{G})$ avec $\mathbf{G} := \mathbf{G}_{lo}$ et $\mathcal{G} := \mathbf{G}_{hi}$, représente un modèle d'une composante réutilisable.

3.4.2.2 Usage de composantes abstraites en contrôle hiérarchique

Ce qui précède ne vaut que pour une seule instance de composante. Pour que cette métaphore de composantes réutilisables soit utile il faut pouvoir assembler des hiérarchies

pyramidales d'instances à la façon illustrée en figure 3.9. Il y a trois *manoeuvres* de conception à envisager.

1. Il doit être possible d'assembler des instances de composantes pour former le comportement libre d'un sous-système (*composition horizontale*).
2. Il faut pouvoir contraindre ce sous-système au comportement désiré par application d'un ensemble de requis de contrôle de portée locale (*contrôle local*).
3. Finalement, le cas échéant, il doit être possible de réabstraire en spécifiant une autre projection à partir du résultat (*composition verticale*).

Chacune de ces trois *manoeuvres* correspond à un opérateur comme illustré en figure 3.10 :

1. le *produit synchrone* pour la *composition horizontale* (figure 3.10 (a)) ;
2. la *composition fonctionnelle de projections* (figure 3.10 (b)) pour la *composition verticale* ;
3. l'*opérateur de synthèse* pour l'élaboration d'un contrôle local (figure 3.10 (c)).

Pour que ce schéma de conception fonctionne, il faut s'assurer que les sous-systèmes *composés* possèdent les propriétés nécessaires à garantir les conditions du théorème 6 de [58] concernant la synthèse non bloquante (au moins dans sa forme affaiblie de l'annexe A). Il est donc nécessaire que chacun des opérateurs de composition ainsi que l'opérateur de synthèse produisent des projections qui préservent

- la forme affaiblie de la consistance du contrôle $\mathcal{C}_{hi}(M) \subseteq \theta(\mathcal{C}_{lo}(L))$;
- la consistance du marquage $\theta^{-1}(M_m) = L_m$;
- la propriété d'observateur ;
- l'absence de couplage du contrôle $\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$.

Dans ces conditions, puisque le produit synchrone et la composition fonctionnelle sont associatifs le schéma de la figure 3.9 devient possible.

Dans ce qui suit il est implicite que l'ensemble des composantes considérées pour la composition sont issues du processus d'abstraction discuté précédemment. En particulier, dans la figure 3.10, toutes les composantes se trouvant à droite du *symbole de définition* (soit := ou ⟵) sont des instances de composantes réutilisables au sens de la section 3.4.2.1, et possèdent donc toutes les propriétés nécessaires à assurer les conditions

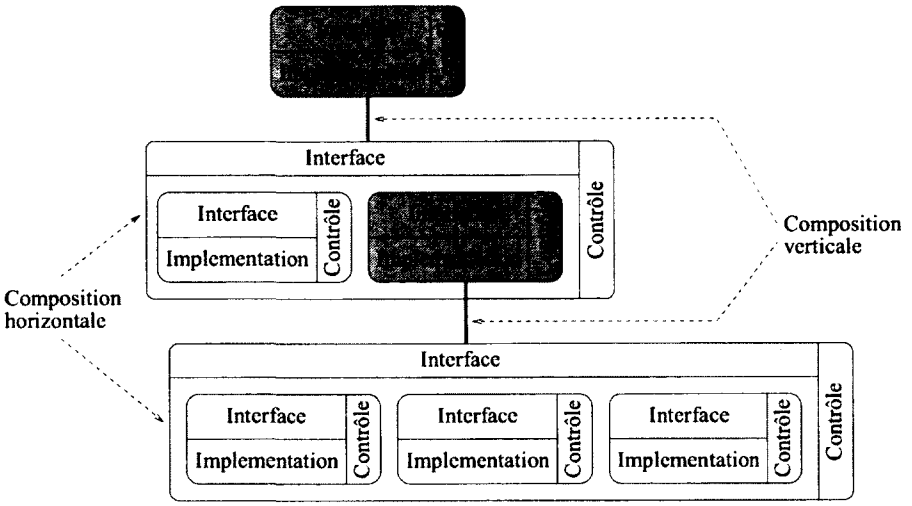


FIGURE 3.9 – Hiérarchie pyramidale de composantes

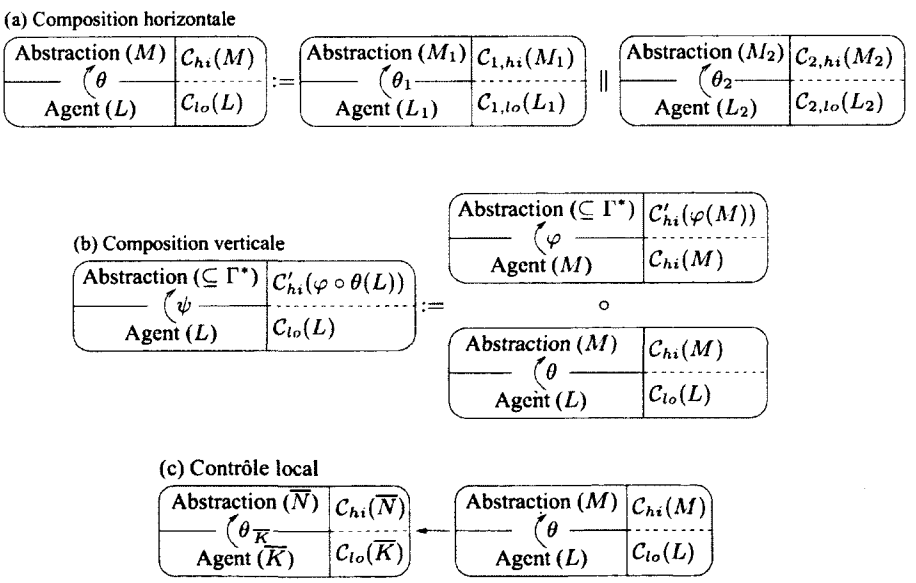


FIGURE 3.10 – Compositions horizontale, verticale et contrôle local

du théorème 6 dans sa forme affaiblie de l'annexe A. De plus le mécanisme d'instanciation suggéré en section 3.1.2.1 a pour but d'assurer que les instances de composantes réutilisables sont *disjointes* deux à deux. Les composantes utilisées en figure 3.10 sont donc sûrement *disjointes* au sens donné à ce terme à la section C.4. On se retrouve précisément dans le contexte détaillé en annexe C et plus particulièrement à la section C.3. Dans ce contexte il doit être clair que la composition synchrone de sous-systèmes munis d'une technologie de contrôle standard résulte en un système lui-même muni d'une telle technologie de contrôle, puisqu'il s'agit du *contexte usuel* en SCT. Similairement, les procédures de synthèse de SCT ne changent rien à la technologie de contrôle du système qui en résulte. Ainsi tous les modèles discutés (composés ou non) possèdent une technologie de contrôle standard et donc une *structure de contrôle standard, localement définissable.*

On considère tout d'abord le cas de la composition horizontale à l'aide de l'opérateur de *produit synchrone* comme illustré en figure 3.10 (a).

On peut montrer que, sous réserve des conditions définies en section C.4, le produit synchrone préserve la propriété d'*absence de délai de contrôle* (proposition C.14) et aussi la propriété d'observateur (proposition C.15). Ainsi, par la proposition C.11, le produit synchrone préserve la propriété de *coïncidence du contrôle* (équation B.2). Ces conditions sont celles du corollaire C.5, ce qui garantit la forme faible de la consistance du contrôle. Aussi, par la proposition C.8, on obtient l'*absence de couplage de contrôle*. Finalement, dans les conditions précitées, le produit synchrone préserve aussi la consistance du marquage (voir la section B.5). Le produit synchrone préserve donc les conditions du théorème 6 dans sa forme affaiblie de l'annexe A.

On considère ensuite le cas de la composition verticale c'est-à-dire la *composition fonctionnelle* de projections comme illustrée en figure 3.10 (b).

Il est utile de noter que, dans la figure 3.10 (b), θ et φ sont toutes deux des projections causales dotées de la propriété d'observateur. D'après le lemme 5 dans [57] (reproduit en section B.6), leur composition fonctionnelle $\psi := \theta \circ \varphi$ est aussi une projection causale dotée de la propriété d'observateur. Aussi θ et φ sont toutes deux construites en respectant la règle de l'équation C.5 et présentent la *coïncidence du contrôle* et donc la *consistance*

du contrôle (version de l'annexe A)

$$\mathcal{C}_{hi}(M) \subseteq \theta(\mathcal{C}_{lo}(L)) \text{ et } \mathcal{C}'_{hi}(\varphi(M)) \subseteq \varphi(\mathcal{C}_{hi}(M))$$

et l'*absence de couplage de contrôle*

$$\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M \text{ et } \varphi_v^{-1} \circ \overline{\kappa}_{\varphi(M)} \leq \kappa_M \circ \varphi^{-1} \circ \overline{\kappa}_{\varphi(M)} .$$

Enfin leur langages marqués sont aussi ajustés pour assurer la *consistance du marquage*

$$\theta^{-1}(\theta(L_m)) = \theta^{-1}(M_m) = L_m \text{ et } \varphi^{-1}(\varphi(M_m)) = M_m .$$

Alors la consistance du contrôle (version affaiblie) est préservée puisque

$$\mathcal{C}'_{hi}(\psi(L)) = \mathcal{C}'_{hi}(\varphi \circ \theta(L)) \subseteq \varphi \circ \theta(\mathcal{C}_{lo}(L)) = \psi(\mathcal{C}_{lo}(L))$$

ainsi que la consistance du marquage

$$\begin{aligned} L_m &= \theta^{-1}(\varphi^{-1}(\varphi(M_m))) \\ &= \theta^{-1}(\varphi^{-1}(\varphi(\theta(L_m)))) \\ &= \theta^{-1}(\varphi^{-1}(\psi(L_m))) \\ &= \psi^{-1}(\psi(L_m)) . \end{aligned}$$

Pour vérifier que l'absence de couplage de contrôle est aussi préservée, on note d'abord que $\kappa_M \circ \varphi^{-1} \circ \overline{\kappa}_{\varphi(M)}$ dénote l'implémentation dans l'*agent* (l'agent M) de la synthèse sur l'abstraction (composante du haut dans la figure 3.10 (b)) d'un certain sous-langage fermé de $\varphi(M)$. Puisqu'il y a absence de couplage de contrôle entre L et M et aussi entre M et $\varphi(M)$, alors

$$\begin{aligned} \theta_v^{-1}(\kappa_M \circ \varphi^{-1} \circ \overline{\kappa}_{\varphi(M)}) &\leq \kappa_L \circ \theta^{-1}(\kappa_M \circ \varphi^{-1} \circ \overline{\kappa}_{\varphi(M)}) \\ \theta_v^{-1}(\varphi^{-1} \circ \overline{\kappa}_{\varphi(M)}) \leq \theta_v^{-1}(\kappa_M \circ \varphi^{-1} \circ \overline{\kappa}_{\varphi(M)}) &\leq \kappa_L \circ \theta^{-1}(\kappa_M \circ \varphi^{-1} \circ \overline{\kappa}_{\varphi(M)}) \end{aligned}$$

et puisque $\kappa_M \circ \varphi^{-1} \circ \overline{\kappa}_{\varphi(M)} \leq \varphi^{-1} \circ \overline{\kappa}_{\varphi(M)}$,

$$\theta_v^{-1}(\varphi_v^{-1} \circ \overline{\kappa}_{\varphi(M)}) \leq \kappa_L \circ \theta^{-1}(\kappa_M \circ \varphi^{-1} \circ \overline{\kappa}_{\varphi(M)}) \leq \kappa_L \circ \theta^{-1}(\varphi^{-1} \circ \overline{\kappa}_{\varphi(M)})$$

et finalement,

$$\begin{aligned}
\theta_v^{-1}(\varphi_v^{-1} \circ \overline{\kappa}_{\varphi(M)}) &\leq \kappa_L \circ \theta^{-1}(\varphi^{-1} \circ \overline{\kappa}_{\varphi(M)}) \\
(\theta_i^{-1} \circ \varphi_i^{-1}) \circ \overline{\kappa}_{\varphi(M)} &\leq \kappa_L \circ (\theta^{-1} \circ \varphi^{-1}) \circ \overline{\kappa}_{\varphi(M)} \\
\psi_v^{-1} \circ \overline{\kappa}_{\varphi(\theta(L))} &\leq \kappa_L \circ \psi^{-1} \circ \overline{\kappa}_{\varphi(\theta(L))} \\
\psi_v^{-1} \circ \overline{\kappa}_{\psi(L)} &\leq \kappa_L \circ \psi^{-1} \circ \overline{\kappa}_{\psi(L)} .
\end{aligned}$$

L'absence de couplage de contrôle est donc préservée elle aussi par la composition fonctionnelle de projections.

À noter que la consistance du contrôle (version faible) et l'absence de couplage de contrôle peuvent aussi être déduites de la préservation de la propriété de coïncidence du contrôle par la composition fonctionnelle de projections puisque

$$\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L) \text{ et } \varphi^{-1}(\mathcal{C}'_{hi}(\varphi(M))) \subseteq \mathcal{C}_{hi}(M)$$

implique que

$$\begin{aligned}
\psi^{-1}(\mathcal{C}'_{hi}(\psi(L))) &= \psi^{-1}(\mathcal{C}'_{hi}(\varphi \circ \theta(L))) \\
&= \psi^{-1}(\mathcal{C}'_{hi}(\varphi(M))) \\
&= \theta^{-1} \circ \varphi^{-1}(\mathcal{C}'_{hi}(\varphi(M))) \\
&\subseteq \theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L) .
\end{aligned}$$

Mais ce qui précède montre qu'il n'est pas nécessaire de présumer la préservation de la coïncidence du contrôle pour obtenir ces deux propriétés. On note aussi que dans la situation de la figure 3.10 (b), aucun contrôle n'est ajouté lors de la réabstraction.

La composition fonctionnelle de projections préserve donc les conditions du théorème 6 dans sa forme affaiblie de l'annexe A.

On considère finalement la situation de la figure 3.10 (c) où on ajoute un contrôle local à un modèle. Dans ce dernier cas, l'abstraction sur laquelle on veut imposer un contrôle (la projection θ) répond déjà à toutes les exigences du théorème 6 (version de l'annexe A), en particulier, θ est un observateur et possède la coïncidence du contrôle.

Soit donc un objectif de contrôle $E \subseteq M_m$ et $N := \kappa_M(E)$ un résultat de synthèse non bloquant dans l'*abstraction*. On a que $K := \theta^{-1}(N)$ est non bloquant dans l'agent (théorème C.10). Par l'argumentation précédant le théorème 6 dans [58], en particulier les propositions 16 et 17, on sait que $\theta_{\overline{K}}$ est un observateur puisque dans ces conditions, la paire (L, θ) est *fortement observable* par rapport à $\overline{K} \in \mathcal{X}$ (annexe A). Puisque K est non bloquant dans l'*agent*, on a que $\overline{K} = \overline{K \cap L_m}$ et $N = \theta(K)$ (par consistance du contrôle et *consistance du marquage*). Ainsi, puisque $K \subseteq L$, on peut affirmer que

$$(\forall s \in L)(\forall t \in T^*)\ \theta(s)t \in \theta(K) \implies (\exists u \in \Sigma^*)\ su \in K \text{ et } \theta(su) = \theta(s)t$$

car $\theta(s)t \in \theta(K) = N$ et K est non bloquant dans l'*agent*. On a donc, par le lemme 2 dans [57], que $\theta_{\overline{K}}$ est un K-observateur. Mais qu'en est-il de la coïncidence du contrôle ?

L'intuition suggère que cette dernière propriété est toujours présente dans le résultat de synthèse $(\overline{K}, \overline{N}, \theta_{\overline{K}})$. On note tout d'abord que $N \in \mathcal{C}_{hi}(M)$, donc $\overline{N} \in \overline{\mathcal{C}}_{hi}(M)$, ainsi $\mathcal{C}_{hi}(\overline{N}) = \mathcal{C}_{hi}(M) \cap \mathcal{P}(\overline{N})$ (propriété 4 des structures de contrôle de la section 2.3.1.2). En suivant un raisonnement similaire on obtient aussi que $\mathcal{C}_{lo}(\overline{K}) = \mathcal{C}_{lo}(L) \cap \mathcal{P}(\overline{K})$ puisque $\overline{K} \in \overline{\mathcal{C}}_{lo}(L)$. Puisqu'on a la *coïncidence du contrôle* entre L et M, on a donc que $\theta^{-1}(\mathcal{C}_{hi}(\overline{N})) \subseteq \theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$. Prenons $R \in \mathcal{C}_{hi}(\overline{N})$ arbitraire. Alors $R \subseteq \overline{N}$ et $\theta^{-1}(R) \subseteq \theta^{-1}(\overline{N}) = \overline{\theta^{-1}(N)} = \overline{K}$ puisque θ est un observateur, et ainsi $\theta^{-1}(R) \in \mathcal{P}(\overline{K})$. Puisque R est arbitraire dans $\mathcal{C}_{hi}(\overline{N})$ on a donc que $\theta^{-1}(\mathcal{C}_{hi}(\overline{N})) \subseteq \mathcal{C}_{lo}(\overline{K})$, c'est-à-dire que la *coïncidence du contrôle* est préservée dans un résultat de synthèse.

Plus généralement on peut conclure qu'une synthèse sur une *abstraction*, conçue d'après la règle de l'équation C.5, donne comme résultat un *observateur du comportement de l'agent en boucle fermée sous le contrôle de* V_K et préserve la *coïncidence du contrôle*.

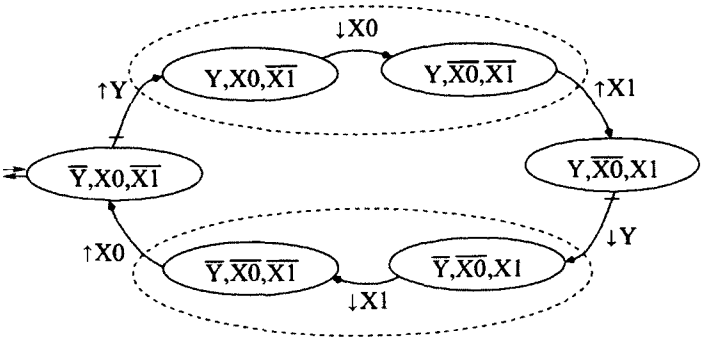
L'imposition d'un contrôle local sur une projection comme illustrée en figure 3.10 (c) préserve donc aussi les conditions du théorème 6 dans sa forme affaiblie de l'annexe A.

Deux points importants se dégagent de ce qui précède. Premièrement, le processus d'abstraction suivant la règle de l'équation C.5 s'avère être extensible tant *verticalement* (abstraction, réabstraction) qu'*horizontalement* (par composition synchrone). À n'importe quel point de la hiérarchie d'abstraction, on peut aussi imposer une loi de contrôle sur un sous-système avant de le composer à nouveau avec d'autres éléments ou de le réabstraire en une autre projection conforme à la règle de l'équation C.5. Cette dernière manoeuvre est utile particulièrement pour *localiser* des requis de contrôle le plus près possible de l'endroit (dans le système) où ils s'appliquent (requis de contrôle de portée purement locale). Deuxièmement, à chaque point de la hiérarchie d'abstraction on obtient une projection conforme à la règle de l'équation C.5. Ceci veut dire en particulier qu'en respectant la règle, ce processus d'abstraction peut être utilisé pour *concevoir des composantes réutilisables*, c'est-à-dire d'autres modèles génériques, à partir d'une combinaison de modèles génériques préexistants ou de modèles exhaustifs.

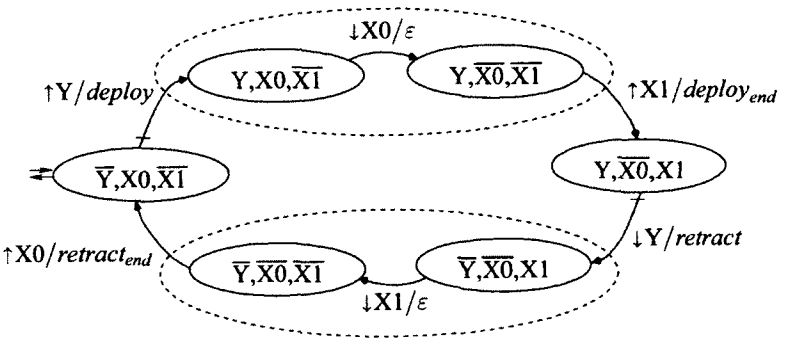
3.4.2.3 Conception de composantes en contrôle hiérarchique

À partir de ce point, pour obtenir le modèle générique d'une composante réutilisable le long des lignes exposées en section 3.1.2.1 (c'est-à-dire une paire $\mathbf{C} = (\mathbf{G}, \mathcal{G})$), il faut définir un *processus d'abstraction*. Ce processus consiste à *concevoir*, à partir de $\mathbf{G}$, une projection θ respectant la règle de l'équation C.5 et dotée de la propriété d'observateur. La figure 3.11 illustre ce processus pour un vérin à commande monostable dont le modèle exhaustif a été préalablement obtenu (figure 3.1). Pour un dispositif physique comme un vérin, le point de départ est nécessairement le modèle exhaustif. Dans le modèle de la figure 3.11 (a), les événements sont des fronts montants et descendants de capteurs (événements incontrôlables) et d'actionneurs (événements contrôlables).

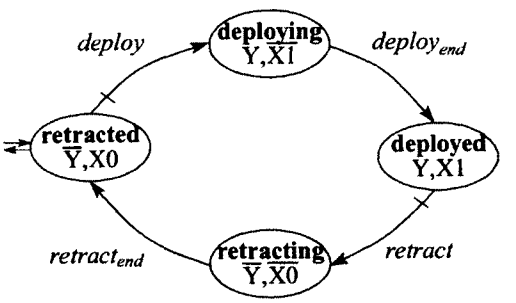
Il faut ensuite faire certaines hypothèses sur les *transitions* qui doivent être projetées et celles qui doivent être effacées. Puisqu'il s'agit d'une projection causale on expose ou on efface des *transitions* et non des événements. On a la possibilité de renommer des événements si nécessaire. On construit donc, en figure 3.11 (b), un modèle de la projection sous la forme d'une machine de Mealy puisque le calcul de la projection utilise l'algorithme de Wong que l'on trouve dans [60]. Lors de la formulation des hypothèses concernant les *transitions* à projeter ou à effacer, on doit veiller à ce que les événements de la projection que



(a) Modèle du materiel conditionné



(b) Modèle de la projection (Machine de Mealy)



(c) Observateur (projection)

FIGURE 3.11 – Abstraction d'un vérin à commande monostable

l'on désire rendre contrôlables ($\tau \in T_c$) n'empruntent que des transitions sur événements contrôlables (c'est-à-dire $\sigma \in \Sigma_c$) dans l'*agent* (ici, dans la machine de Mealy conçue à partir du modèle exhaustif).

Par la suite on utilise l'algorithme de Wong pour calculer la quasi-congruence la plus grossière qui soit plus fine que le modèle de la projection spécifiée par le modèle de la figure 3.11 (b) résultant de nos hypothèses. L'algorithme nous retourne l'automate quotient (section 2.8.1.2) le plus grossier qui soit conforme à nos hypothèses. Éventuellement l'algorithme doit cependant ajouter de nouvelles hypothèses sous forme d'autres transitions à exposer. On peut accepter le résultat de l'algorithme ou reformuler d'autres hypothèses sur la base de ce que l'algorithme nous suggère. Éventuellement, en raffinant nos hypothèses de la sorte, on obtient une projection d'observateur, c'est-à-dire que l'algorithme de Wong n'ajoute plus aucune hypothèse et nous fournit le quotient le plus grossier correspondant à nos hypothèses.

La figure 3.11 (c) illustre l'AFD quotient obtenu de l'algorithme de Wong à partir des hypothèses faites en figure 3.11 (b). Bien sûr ce modèle est simple. On peut vérifier *par inspection* que la proposition 7 de [58] est vérifiée pour ce quotient et donc qu'il s'agit bel et bien d'un observateur. Ce quotient correspond d'ailleurs *presque* à un *observateur naturel* (voir [15]), mais pas tout à fait car les événements projetés ont été *renommés*. Mais à cette différence près, il s'agit d'un observateur naturel.

Bien sûr le processus est heuristique. Il ne saurait en être autrement car à moins d'encoder ces heuristiques dans la procédure de réétiquetage de l'algorithme de Wong, l'algorithme lui-même ne peut fournir que des suggestions. Aussi la convergence de l'algorithme repose sur le fait que la procédure de réétiquetage utilise (pour ses suggestions) de nouvelles étiquettes d'événements qui sont toutes différentes de celles qui existent déjà. Modifier la procédure de réétiquetage devrait tenir compte des exigences pour la convergence de l'algorithme et il semble peu probable que l'on puisse faire mieux que ce qui existe déjà dans l'algorithme de Wong. D'ailleurs, vu l'étendue des possibilités pour le réétiquetage, il semble préférable de laisser la tâche de faire ce genre d'hypothèses au concepteur de la projection. Ce dernier est mieux à même de décider de critères comme la sémantique des noms d'étiquettes. L'expérience semble suggérer que les hypothèses supplémentaires faites par l'algorithme de Wong suffisent à guider le concepteur de la projection efficacement dans le

processus.

Un processus heuristique comme celui illustré en figure 3.11 peut sembler limité et coûteux. Il est limité parce que la taille des modèles doit généralement permettre l'analyse par *inspection* visuelle. Il est coûteux car sa complexité est de l'ordre de $O(k^4 n)$ où k est la taille de l'espace des états de la machine de Mealy et n le nombre de ses transitions (voir [60]). Si on ajoute à cela que plusieurs utilisations de l'algorithme sont généralement nécessaires pour trouver un observateur, le coût peut sembler prohibitif.

On peut opposer au premier argument que, tout au moins pour l'abstraction à partir de dispositifs physiques, les modèles exhaustifs impliqués sont généralement de taille modeste. C'est certainement le cas dans les usines modulaires.

Pour l'argument concernant les coûts, on doit remarquer que les bénéfices principaux de ce type de processus d'abstraction sont la réutilisation de spécification (sous forme de composantes réutilisables) et l'ensemble des garanties formelles qu'il procure. La réutilisation de composantes par *instanciation* dans divers problèmes de contrôle procure une façon d'amortir les coûts de conception. Aussi, une fois franchie la barrière de la conception du modèle exhaustif (un processus *ad hoc* pour lequel on ne dispose d'aucune garantie formelle sauf peut-être sa validation statistique), ce type de processus nous procure une façon de passer du domaine informel (*ad hoc*) au domaine formel de SCT et d'y rester. On dispose ainsi des garanties formelles (synthèses contrôlables et non bloquantes) que SCT procure.

On peut ajouter qu'il est aussi très coûteux d'obtenir des modèles exhaustifs *adéquatement vérifiés* pour les dispositifs physiques et que pour l'instant c'est la seule alternative qui existe. Or l'élaboration d'une composante réutilisable pour un tel dispositif (éventuellement avec plusieurs éléments d'interface $\mathcal{G}$) permet au moins de récupérer le travail investi dans l'élaboration du modèle exhaustif lui-même.

3.4.3 Sommaire d'analyse

Par rapport aux critères fixés en section 3.1, on peut dire que le contrôle hiérarchique s'accommode bien de toutes les limitations sauf celles concernant les solutions de synthèse de type SFBC (L4). La limitation à des sous-systèmes *disjoints* (L1) devient en fait, pour

l'abstraction le long des lignes de la règle de l'équation C.5, un préalable.

Pour procurer des solutions de synthèse de type SFBC, on peut toujours avoir recours à une modélisation de système par STS. Mais dans ce cas le système modélisé devrait être un système *plat*, c'est-à-dire sans structure verticale. Ce genre de modèle est composé d'un seul super-état ET x_0 à la racine ($T(x_0) = and$), et chacune de ses *cellules* (des super-états OU) est associée directement à l'instance d'une paire $(\mathbf{G}_{lo}, \mathcal{G})$ (l'instance d'un modèle générique conçu selon le processus illustré en section 3.4.2.3). Le comportement libre du système *abstrait* est alors équivalent à

$$\mathcal{G} = \|_{i \leq n} \mathcal{G}_i$$

pour $i \in \{1, ..., n\}$. En contrôle hiérarchique ce modèle *abstrait* est un observateur du système *concret* formé du produit

$$\mathbf{G}_{lo} = \|_{i \leq n} \mathbf{G}_{i,lo}$$

des *agents*, pour $i \in \{1, ..., n\}$, et dotée de la *coïncidence du contrôle*. Ce type de modèle de système est la version STS d'un modèle de comportement libre dans le contexte usuel de SCT (système *plat*).

La figure 3.3 fournit un exemple d'un modèle de STS sans structure verticale, si on suppose que les holons x et y dans l'élément **Flèche** ne constituent qu'un seul état *opaque* (comme pour le vérin dans la figure 3.11 (c)). On peut y modéliser les requis de contrôle à l'aide d'une spécification similaire à celle de la figure 3.4 (par famille d'états proscrits) et, éventuellement, par modélisation explicite d'éléments de mémoire (la pratique usuelle en contrôle de STS). De cette façon on récupère les avantages liés à une synthèse symbolique livrant une SFBC et à une modélisation explicite de la mémoire.

L'usage d'une modélisation par STS (même avec un système sans structure verticale comme précédemment) procure en plus un avantage. On se rappelle que l'algorithme de Wong livre, comme *abstraction*, un AFD quotient (section 2.8.1.2) de la structure de tran-

sition de G, la machine de Mealy. Dans le quotient

$$\overline{\mathbf{G}} := \widehat{\mathbf{G}}/(T, \rho_{\mathbf{G}}) = (Y, T \cup \{\tau_0\}, \eta, y_0, Y_m)$$

$\rho_{\mathbf{G}}$ représente une *partition* de l'espace des états de la *structure de transition* de la machine de Mealy, notée $\mathbf{G_{ts}}$. Dans notre cas on a que $\mathbf{G_{ts}} = \mathbf{G}_{lo}$. Les composantes réutilisables obtenues du processus d'abstraction de la section 3.4.2.3 sont donc des paires $(\mathbf{G_{ts}}, \mathcal{G})$ où $\mathcal{G}$ est un observateur possédant la *coïncidence du contrôle*. Chaque état de $\mathcal{G}$ représente un élément de la partition $\rho_{\mathbf{G}}$ donc un ensemble des états de $\mathbf{G_{ts}}$ *disjoints* des autres éléments de $\rho_{\mathbf{G}}$.

En synthèse de STS, pour le produit

$$\mathcal{G} = \|_{i \leq n} \mathcal{G}_i$$

($i \in \{1, \ldots n\}$), le résultat est donné sous forme d'un ensemble de prédicats (f_σ pour $\sigma \in \Sigma_c$) représentant des fonctions caractéristiques sur $\mathcal{G}$ (des indicatrices de familles d'états). La relation entre les états des *abstractions* composant le système, $\mathcal{G}_i$, et de leurs *agents* respectifs, $\mathbf{G}_{i,\mathbf{ts}}$, à travers $\rho_{\mathbf{G}_i}$ procure donc un support utile pour *traduire* le résultat d'une synthèse sur le modèle abstrait en un contrôle équivalent sur le modèle concret. On a ainsi un support éventuel pour l'*implémentation* d'un résultat de synthèse sur l'abstraction d'un problème de contrôle (critère C8).

Ainsi on peut dire que, sous provision

- de respecter la règle de l'équation C.5 pour l'abstraction de composantes réutilisables ;
- d'utiliser le processus d'abstraction décrit en section 3.4.2.3 ;
- d'utiliser des modèles de STS sans structure verticale (comme décrit précédemment) pour la modélisation de problèmes de contrôle ;

le contrôle hiérarchique supporte la conception d'une métaphore de composantes réutilisables selon les lignes de la section 3.1.2.1. Les critères définis en section 3.1 semblent tous adéquatement remplis.

L'élément de modularisation est une paire $(\mathbf{G}, \mathcal{G})$ où $\mathcal{G}$ est une *abstraction* de l'*agent* G

conçue en respectant la règle de l'équation C.5. L'interface est constituée de l'élément $\mathcal{G}$, elle procure une encapsulation opaque de la dynamique de l'*agent* et simplifie la synthèse. Les procédures de calcul d'observateur de Wong constituent un support à la dérivation d'interfaces (ici, l'*abstraction*). Le processus d'abstraction peut être répété verticalement, et utilisé pour la conception de composantes réutilisables autant que pour la formulation de problèmes de contrôle. En utilisant une modélisation par STS sans structure verticale (comme discuté précédemment), les procédures de synthèse du contrôle de STS sont adéquates avec cette métaphore de composantes réutilisables. Ces procédures livrent une SFBC comme résultat de synthèse et utilisent des BDD (méthodes symboliques) pour leurs calculs. Comme mentionné précédemment, l'usage de l'algorithme de Wong et d'un résultat de synthèse de type SFBC procurent un support substantiel à l'implémentation des résultats de synthèse. L'encapsulation opaque entraîne une perte d'optimalité dans la synthèse, mais cette perte d'optimalité peut, en pratique, être considérée raisonnable.

3.5 Solution proposée

Dans les sections qui précèdent on a tenté d'identifier, dans l'état courant du développement de la théorie du contrôle supervisé, un contexte dans lequel il est possible de formuler des problèmes de contrôle et de les résoudre en utilisant une méthode de conception par composantes réutilisables. La méthode doit couvrir le problème global, de la formulation de problèmes de contrôle à l'implémentation de la solution obtenue par synthèse. L'un des éléments clefs est bien sûr la formulation d'une métaphore de composantes réutilisables adéquate. Un scénario basé sur l'instanciation de modèles génériques, similaires à la notion de *classes* en programmation par objets, est proposé en section 3.1.2.1. Un mécanisme d'instanciation, correspondant à une forme de déclaration de variables au sein d'un modèle de problème de contrôle, y est aussi proposé. Chaque variante hiérarchique de SCT a alors été examinée sous l'angle de ce scénario.

On peut se rendre compte que, malgré sa notation particulièrement expressive et compacte (une forme des *statecharts* de Harel), le contrôle de STS présente un problème lorsque l'on tente d'y identifier un élément de modularisation pouvant supporter une métaphore de composantes réutilisables. Le holon, l'élément de base de la dynamique en contrôle de

STS, semble ne pas être adéquat pour la tâche. Le holon semble en effet être une expression *d'instances de comportements* d'un sous-système au sein d'un système global. L'ensemble de ces instances de comportements dans un système semble ne pas pouvoir être associé aisément à une composante physique cyclique comme dans l'exemple du vérin en figure 3.7. Seule l'expression d'un *holon en forme fondamentale* (c'est-à-dire un holon sans structure externe) est adéquate pour l'expression de composantes réutilisables (modèles génériques) et ceci appauvrit beaucoup trop le formalisme de modélisation par STS. En effet, un modèle de STS n'est alors qu'un système sans structure verticale et toute la puissance d'expressivité des STS disparaît d'un coup. La variante demeure particulièrement attrayante cependant à cause de sa modélisation explicite de la mémoire (sous forme de contraintes de contrôle) et de ses procédures de synthèse produisant une loi de contrôle de type SFBC et utilisant des méthodes symboliques en synthèse.

En ce qui concerne le contrôle hiérarchique par interface (HISC), bien qu'il soit clair que l'on puisse formuler une métaphore de composantes réutilisables adéquate, certaines limitations de cette variante la rendent moins intéressante que le contrôle hiérarchique de Wong. Premièrement, même telle que formulée en section 3.3.2, cette métaphore n'est pas extensible sur l'axe vertical parce que HISC est essentiellement une méthode limitée à deux niveaux actuellement. L'appareillage théorique n'existe tout simplement pas. Ceci limite sérieusement la conception de composantes réutilisables puisqu'elles ne peuvent pas faire usage d'autres composantes réutilisables préexistantes pour leur conception. Dans ce derniers cas, la réutilisation se limite à une réutilisation *au niveau du code source* (copier-coller ou *rote-reuse* en anglais). Ainsi, bien que l'on ne puisse dire qu'il n'y en a pas, il y a peu de support formel pour la conception de composantes réutilisables. L'usage de *projections naturelles* pour les interfaces HISC présente aussi certains problèmes. La conception d'une projection naturelle adéquate comme interface en paires commandes-réponses (les interfaces HISC) relève largement de l'intuition et de certaines techniques comme celle des *événements synthétiques* illustrée en figure 3.8, et aucun algorithme n'est suggéré. Certaines techniques, comme celle des *événements synthétiques*, demeurent cependant très utiles. Elles peuvent d'ailleurs aussi être utilisées en contrôle hiérarchique (contexte de Wong et Wonham).

De ce qui précède il ressort que la variante du contrôle hiérarchique due à Wong et

Wonham (section 2.3) semble être la plus adéquate pour formuler une métaphore de composantes réutilisables sur le modèle de la section 3.1.2.1. En particulier la possibilité de répéter le processus d'abstraction le long de l'axe vertical et de concevoir de nouvelles abstractions à partir d'une composition d'abstractions existantes est particulièrement attrayante. Cette caractéristique rend la distinction entre sous-système et composante réutilisable presque inexistante. En effet un sous-système peut très bien être *programmé* comme une composante générique et instancié lors de la formulation d'un problème de contrôle. La conception de composantes réutilisables revêt alors presque le même aspect que la notion de *classe* en programmation par objets.

Le contrôle hiérarchique procure un bon support formel pour la dérivation d'interfaces (l'algorithme de Wong). Aussi, le contrôle hiérarchique ne dépend d'aucune procédure de synthèse particulière. Il peut utiliser aussi bien celles du contexte original de SCT (Ramadge et Wonham) que les procédures de synthèse du contrôle de STS. C'est d'ailleurs pourquoi la section 3.4.3 suggère l'usage de composantes réutilisables, obtenues du contrôle hiérarchique, dans un contexte de modélisation de problèmes de contrôle par STS. Même sans structure verticale un tel modèle présente plusieurs caractéristiques intéressantes. Les composantes réutilisables issues du contrôle hiérarchique peuvent réduire significativement l'espace des états du modèle du problème de contrôle. Alliée à l'usage de BDD (méthodes symboliques) en synthèse, cette réduction de l'espace des états ne peut que diminuer la probabilité d'explosion de l'espace des états et aider à aborder des problèmes de contrôle de taille réaliste. La structure verticale se trouve encapsulée de façon opaque dans les composantes réutilisables et les sous-systèmes, elle n'y est donc pas absente. La modélisation explicite de la mémoire (sous forme de contraintes) ainsi que l'algorithme de Wong, dont les *modèles d'observateurs* sont des quotients d'AFD, procurent un support intéressant pour l'implémentation des solutions de synthèse. Enfin la méthode de spécification des requis de contrôle, par familles d'états proscrits (un ensemble de ST ou de prédicats), rendent les requis de contrôle plus aisés à formuler et plus faciles à relier à leur expression formelle.

On peut affirmer en conclusion que ce qui se dégage de cette analyse est que la meilleure recommandation semble être de procéder avec un hybride. Cet hybride utilise le contrôle hiérarchique, le long des lignes directrices exposées en section 3.4.2.3, pour la conception

de composantes réutilisables. En plus, l'hybride utilise une modélisation de problèmes de contrôle par STS sans structure verticale pour l'accès aux techniques de spécification des requis de contrôle caractéristique du contrôle de STS, et la synthèse utilisant des méthodes symboliques (des BDD) et produisant une loi de contrôle de type SFBC. Cette méthode n'est sûrement pas une panacée mais elle promet d'être intéressante à explorer et semble pouvoir livrer des résultats substantiels. Le chapitre 4 procède à une description détaillée d'un tel hybride et le chapitre 5 donne un exemple substantiel de son usage pour la conception de contrôleurs.

Chapitre 4

Modélisation, composantes, sous-systèmes et problèmes de contrôle

Le chapitre 3 préconise une conception de composantes réutilisables basée sur le paradigme *modèle + abstraction* dans le contexte du contrôle hiérarchique de Wong et Wonham (voir la section 3.4). Le chapitre 3 propose en fait de concevoir des composantes réutilisables selon les principes du contrôle hiérarchique mais d'utiliser une modélisation de systèmes utilisant les STS (voir la section 2.7). Dans cette approche, les modèles de systèmes doivent cependant être limités à des STS *sans structure verticale* (voir la section 3.4.3 et la figure 2.6 pour une illustration). De cette façon on bénéficie de procédures de synthèse utilisant des méthodes symboliques (usage de BDD, voir la section 2.6), d'un résultat de synthèse sous forme de DSFBC (voir la section 2.5) et de la modélisation explicite des éléments de mémoire dans le système pour l'élaboration de contraintes (requis de contrôle). Une composante réutilisable peut alors être représentée par une paire $C = (G, \overline{G})$ où G est une machine de Mealy déterministe (voir la section 2.3.3.2) et $\overline{G}$ un AFD quotient (voir la section 2.8.1.2) établi à partir de G en utilisant l'algorithme donné par Wong et Wonham dans [60] en prenant soin de respecter la règle de l'équation C.5. Il s'agit essentiellement d'une variation du schéma proposé en section 3.1.2.1 pour modéliser les composantes réutilisables, puisqu'une machine de Mealy déterministe G est toujours associée à G_{ts} (un AFD définissant sa structure de transition).

Dans une composante réutilisable $\mathbf{C} = (\mathbf{G}, \overline{\mathbf{G}})$ telle que décrite précédemment, la machine de Mealy

$$\mathbf{G} = (Q, \Sigma, T \cup \{\varepsilon\}, \delta, \lambda, q_0, Q_m) \tag{4.1}$$

est un modèle détaillé de la paire (L, θ) (la *mécanique* de la projection), et l'AFD quotient

$$\overline{\mathbf{G}} = (Y, T, \eta, y_0, Y_m) \tag{4.2}$$

est un modèle de $M := \theta(L) \subseteq T^*$ où $L := L(\mathbf{G_{ts}})$. Aussi, puisque $\overline{\mathbf{G}}$ est calculé à partir de $\mathbf{G}$ à l'aide de l'algorithme de Wong en respectant la règle de l'équation C.5, on obtient que $\theta : L \rightarrow M$ est une projection causale dotée de la propriété d'observateur (voir la section 2.3.3) et de la coïncidence du contrôle (voir la section B.3). En supposant alors que le sous-système $\mathbf{G_{ts}}$ dispose a priori de la technologie de contrôle $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$ (la technologie de contrôle standard), on obtient que l'abstraction $\overline{\mathbf{G}}$ est aussi munie d'une technologie de contrôle standard $T = T_c \dot{\cup} T_u$.

On peut d'ores et déjà remarquer qu'en posant $\mathbf{G}_{lo} := \mathbf{G_{ts}}$ et $\mathbf{G}_{hi} := \overline{\mathbf{G}}$ on établit le contexte du contrôle hiérarchique (voir la section 2.3) dans les conditions décrites en annexe C. On dispose donc de la consistance hiérarchique restreinte entre le sous-système $\mathbf{G_{ts}}$ et son abstraction $\overline{\mathbf{G}}$ et, plus généralement, des résultats sur la synthèse non bloquante de l'annexe A et des résultats de l'annexe C sur la composition synchrone et la réabstraction d'observateurs. On peut donc considérer que dans la paire $\mathbf{C} = (\mathbf{G}, \overline{\mathbf{G}})$, le modèle de la projection $\mathbf{G}$ est construit autour de l'*implémentation* d'un sous-système ($\mathbf{G_{ts}}$) dont l'abstraction $\overline{\mathbf{G}}$ représente une *interface*. En supposant un mécanisme d'instanciation adéquat comme celui proposé en section 3.1.2.1, on peut former une ou plusieurs instances de $\mathbf{C} = (\mathbf{G}, \overline{\mathbf{G}})$ qui est alors considéré comme un modèle *générique* pour la construction d'instances. En utilisant la notation de la section 3.1.2.1, toute instance C de $\mathbf{C} = (\mathbf{G}, \overline{\mathbf{G}})$, définit deux structures de transition $C.\mathbf{G}$ et $C.\overline{\mathbf{G}}$ respectivement isomorphes à $\mathbf{G}$ et $\overline{\mathbf{G}}$ aux étiquettes d'*objets formels* près (ensembles, fonctions de transition, étiquettes d'événements et d'états). Ainsi, dans le contexte du contrôle hiérarchique on peut poser $\mathbf{G}_{lo} := C.\mathbf{G_{ts}}$ et $\mathbf{G}_{hi} := C.\overline{\mathbf{G}}$ avec les mêmes garanties formelles que pour la paire $(\mathbf{G_{ts}}, \overline{\mathbf{G}})$ du modèle générique.

4.1 Modélisation de sous-système

On peut légitimement se poser la question de savoir s'il suffit de spécifier le modèle de chacune des deux structures de transition d'une paire $\mathbf{C} = (\mathbf{G}, \overline{\mathbf{G}})$ pour spécifier adéquatement une composante réutilisable. Dans ce qui précède on voit apparaître l'élément $\mathbf{G_{ts}}$, la structure de transition du modèle de la paire (L, θ), la projection. Or $\mathbf{G_{ts}}$ est un sous-système à part entière ayant éventuellement fait l'objet d'une synthèse (voir la section 3.4.2.2). Autrement dit $\mathbf{G_{ts}}$ représente la solution d'un sous-problème de contrôle. Une spécification de composante réutilisable $\mathbf{C} = (\mathbf{G}, \overline{\mathbf{G}})$ doit donc nécessairement *documenter* la solution du sous-problème de contrôle dont $\mathbf{G_{ts}}$ exprime la solution.

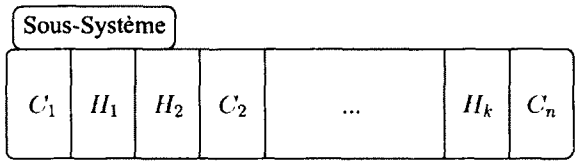
4.1.1 Éléments d'un sous-système

La figure 4.1 illustre la structure générale d'un sous-système conçu à partir de composantes réutilisables du type $\mathbf{C} = (\mathbf{G}, \overline{\mathbf{G}})$ comme proposé précédemment. Un tel sous-système est généralement composé d'une ou plusieurs instances de composantes réutilisables auxquelles on ajoute éventuellement des éléments de mémoire pour la spécification des requis du contrôle.

4.1.1.1 Modèle général d'un sous-système

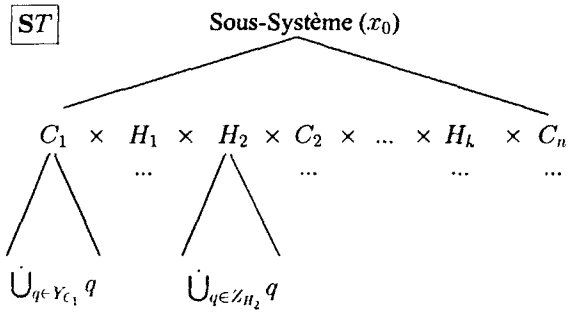
Le modèle de la figure 4.1 (A) est un STS sans structure verticale (**Sous-Système**$_{\mathbf{STS}}$). Il n'est formé que d'un seul super-état ET à la racine (x_0), dénoté «Sous-Système», dont chaque *cellule* est un super-état OU superposé à un AFD considéré comme un *holon* dont la structure externe est vide. Chaque cellule est donc un sous-STS en mode fondamental (voir la section 3.2). Le *holon* superposé à un super-état OU représentant un élément de mémoire M_j ($j \in \{1, \ldots, k\}$) est tout simplement l'AFD correspondant à cet élément tel que défini dans l'*environnement* du sous-système et dénoté $\mathbf{H}_{M_j}$. Le *holon* superposé à un super-état OU représentant une *instance* de composante réutilisable est une *instance du modèle abstrait* $\overline{\mathbf{G}}_{C_i}$ ($i \in \{1, \ldots, n\}$) de ce *type* de composante réutilisable tel que défini dans l'*environnement* du sous-système. Le *type* d'une *instance* de composante réutilisable C_i provient bien sûr d'un *modèle générique* $\mathbf{C}_\alpha = (\mathbf{G}_\alpha, \overline{\mathbf{G}}_\alpha)$ pour $\alpha \in A$ un ensemble

A) **Sous-Système$_{\mathbf{STS}}$** $= (\mathbf{ST}, \mathcal{H}, \Sigma, \Delta, \mathbf{ST}_0, \mathbf{ST}_m)$



$\mathcal{H}^{C_i} \sim \overline{\mathbf{G}}_{C_i} = (Y_{C_i}, T_{C_i}, \eta_{C_i}, y_{C_i 0}, Y_{C_i m})$

$\mathcal{H}^{H_j} \sim \mathbf{H}_{H_j} = (Z_{H_j}, W_{H_j}, \zeta_{H_j}, z_{H_j 0}, Z_{H_j m})$



$\mathcal{H} := \bigcup_{i \in \{1 \quad n\}} \mathcal{H}^{C_i} \cup \bigcup_{j \in \{1 \quad k\}} \mathcal{H}^{H_j}$

$\Sigma := \bigcup_{i \in \{1 \quad n\}} T_{C_i}$

B) **Sous-Système$_{\mathbf{AFD}}$** $:= \overline{\mathbf{G}}_{C_1} \parallel \mathbf{H}_1 \parallel \mathbf{H}_2 \parallel \overline{\mathbf{G}}_{C_2} \parallel ... \parallel \mathbf{H}_k \parallel \overline{\mathbf{G}}_{C_n} = (Q, \Sigma, \delta, q_0, Q_m)$

$q \in Q \subseteq Y_{C_1} \times Z_{H_1} \times Z_{H_2} \times Y_{C_2} \times ... \times Z_{H_k} \times Y_{C_n}$

FIGURE 4.1 – Modèle d'un sous-système

d'index (par exemple une librairie de composantes réutilisables).

Le STS de la figure 4.1 (A) est *bien formé* et *déterministe* par construction (voir [39] à la section 2.3). Les instances de composantes réutilisables C_i, $i \in \{1, ..., n\}$, sont disjointes deux à deux, elles ont donc toutes des espaces d'états distincts et des alphabets d'événements disjoints des autres instances. L'ensemble des *holons* $\mathcal{H}^{C_i}$ définit ainsi le *comportement libre* du sous-système. L'ensemble des *mémoires*, H_j, $j \in \{1, ..., k\}$, est appelé à supporter la description du problème de contrôle, c'est-à-dire la *spécification du contrôle*. Sans perte de généralité on peut supposer que les éléments de mémoire sont définis avec des espaces d'états disjoints de tous les autres *holons* du modèle. La mémoire n'a pas de dynamique propre donc $W_{H_j} \subseteq \Sigma$ pour $j \in \{1, ..., k\}$ où $\Sigma := \bigcup_{i \in \{1, ..., n\}} T_{C_i}$, c'est-à-dire que l'activité de l'ensemble des mémoires est synchronisée sur la dynamique du *comportement libre* du sous-système. Le modèle respecte donc la *condition de couplage local* (voir la section 2.7.1.3).

D'après la définition 2.9 de [39], n'importe quelle sous-arborescence d'états $\mathrm{ST}_\alpha \leq \mathrm{ST}$ peut être décrite complètement par ce qu'il est convenu d'appeler l'*ensemble des états actifs* du sous-système :

$$\mathcal{V}_{\mathrm{ST}_\alpha} := \bigcup_{z \in Z} \mathcal{V}_{\mathrm{ST}_\alpha}(z)$$

où Z est l'ensemble des super-états OU comportant des *états actifs* sur ST_α. Dans un STS sans structure verticale (un sous-système *plat*) comme celui de la figure 4.1, on vérifie aisément que si $\mathcal{V}_{\mathrm{ST}_\alpha}(C_i) \neq \emptyset$ alors $\mathcal{V}_{\mathrm{ST}_\alpha}(C_i) \subset Y_{C_i}$ pour $i \in \{1, ..., n\}$, une inclusion stricte. En effet d'après la définition 2.9 de [39], pour i fixé, si le sous-arbre ST^{C_i} figure intégralement dans ST_α, alors son ensemble d'états actifs est vide, $\mathcal{V}_{\mathrm{ST}_\alpha}(C_i) = \emptyset$. On a une situation similaire concernant les mémoires H_j pour $j \in \{1, ..., k\}$. Ainsi toute arborescence d'état élémentaire $b \in \mathcal{B}(\mathrm{ST})$ (une *configuration* du sous-système) peut être exprimée par :

$$\mathcal{V}_b := \left(\bigcup_{i \in \{1 \ ..n\}} \mathcal{V}_b(C_i) \right) \cup \left(\bigcup_{j \in \{1, \ ..k\}} \mathcal{V}_b(H_j) \right) .$$

Or d'après la définition de $b \in \mathcal{B}(\mathrm{ST})$ (voir la définition 2.12 dans [39]), dans un sous-système tel que celui de la figure 4.1 (un sous-système *plat*), les ensembles $\mathcal{V}_b(C_i)$ ne

contiennent qu'un et un seul état actif et il en va de même pour les ensembles $\mathcal{V}_b(H_j)$. Alors il existe une bijection $h : \mathcal{B}(ST) \rightarrow Q$, pour ST et Q tels q'illustrés respectivement en (A) et (B) de la figure 4.1. Ainsi, pour $b \in \mathcal{B}(ST)$ et $\sigma \in \Sigma$, les fonctions Δ et δ sont liées par

$$\delta(h(b), \sigma)! \iff \Delta(b, \sigma) \neq \emptyset$$

où $\emptyset$ dénote l'arborescence d'état vide, et alors

$$\delta(h(b), \sigma) = h(\Delta(b, \sigma)) .$$

Autrement dit, la fonction Δ est aussi *juste* par construction (voir à la section 2.7.1.3 et [39] à la section 2.3.1). Un STS défini comme à la figure 4.1 (A) est ainsi un modèle parfaitement équivalent à celui d'un sous-système obtenu par composition synchrone tel qu'illustré en figure 4.1 (B) où

$$Q := \bigcup_{b \in \mathcal{B}(ST)} h(b) .$$

Cette structure de sous-système (le sous-système *plat*) est d'ailleurs utilisée dans deux examples de problèmes de contrôle provenant de [39] à la section 4.5, le problème *Small Factory* et le problème *Transfer Line*, illustrés respectivement par les figures 4.26 et 4.31 de [39].

L'expression de $q \in Q$ sous forme d'un l-uplet ($l := n + k$) constitue donc une représentation adéquate de l'élément $b \in \mathcal{B}(ST)$ qui lui correspond dans ST puisque ce l-uplet dénote l'ensemble des états actifs de b *à une différence de notation près*. Il est alors utile de définir le l-uplet

$$v_{Sous\text{-}Système} = (v_{C_1}, ..., v_{C_n}, v_{H_1}, ..., v_{H_k}) \tag{4.3}$$

comme *vecteur d'état* pour le sous-système, où v_{C_i} ($i \in \{1, ..., n\}$) et v_{H_j} ($j \in \{1, ..., k\}$) sont les *variables d'état* qui correspondent aux éléments qui composent le sous-système (instances de composantes réutilisables et éléments de mémoire).

On constate que l'ordre des variables d'état de l'équation 4.3 diffère de celui donné pour le STS **Sous-Système$_{STS}$** de la figure 4.1 (A). La définition de $v_{Sous\text{-}Système}$ a pour but

d'assigner un ordre précis à l'énumération des variables d'état dans le l-uplet de façon à abréger la notation, il n'est pas nécessaire de respecter l'ordre donné par le STS qui lui est pertinent à la synthèse. Une *configuration* du vecteur d'état du sous-système ($v_{\textit{Sous-Système}}$) correspond ainsi à une *configuration* du STS **Sous-Système**$_{\textbf{STS}}$ *à l'ordre des variables près*. Bien sûr l'ordre des variables dans **Sous-Système**$_{\textbf{STS}}$ correspond à la synthèse et doit être spécifié comme l'illustre la figure 4.1 (A).

4.1.1.2 Prédicats, spécification de contrôle et résultat de synthèse

Le sous-système doit habituellement être contraint, par spécification et synthèse, à un comportement contrôlé. La figure 4.2 illustre les éléments de ce problème de contrôle. On y détecte deux éléments importants : la *spécification de contrôle* dénotée $\mathcal{S}$, et le résultat de synthèse, une loi de contrôle f de type SFBC.

D'après Ma et Wonham [39] (section 4.1), la spécification d'un problème de contrôle prend la forme d'une paire formée d'un STS et d'un ensemble $\mathcal{S} \subseteq ST(\mathbf{ST})$ de sous-arborescences d'états *illégales* pour le sous-système. Les *configurations interdites* du sous-système (celles que le sous-système doit éviter par contrôle) sont alors données par

$$B_{\mathcal{S}} := \bigcup_{S \in \mathcal{S}} \mathcal{B}(S)$$

et le *prédicat* de spécification,

$$P_{\mathcal{S}} := \bigvee_{S \in \mathcal{S}} \Theta(S) = \bigvee_{b \in B_{\mathcal{S}}} \Theta(b)$$

une fonction indicatrice de $B_{\mathcal{S}}$, est obtenu par application de la fonction Θ (voir [39] section 4.2.1 et la section 2.7.3). Alors $b \models P_{\mathcal{S}} \iff b \in B_{\mathcal{S}}$ pour $b \in \mathcal{B}(\mathbf{ST})$. Naturellement il est possible de former

$$Q_{\mathcal{S}} := \bigcup_{b \in B_{\mathcal{S}}} h(b)$$

et alors

$$q \models P_{\mathcal{S}} \iff h^{-1}(q) \in B_{\mathcal{S}} \iff q \in Q_{\mathcal{S}}$$

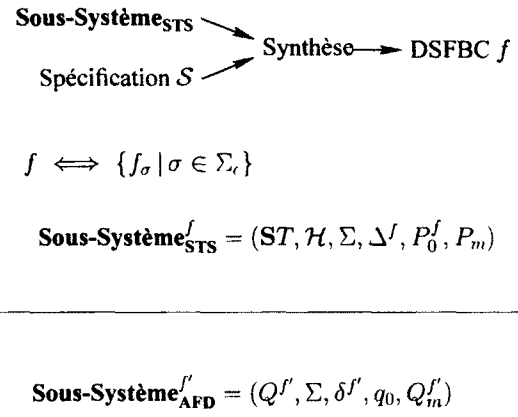


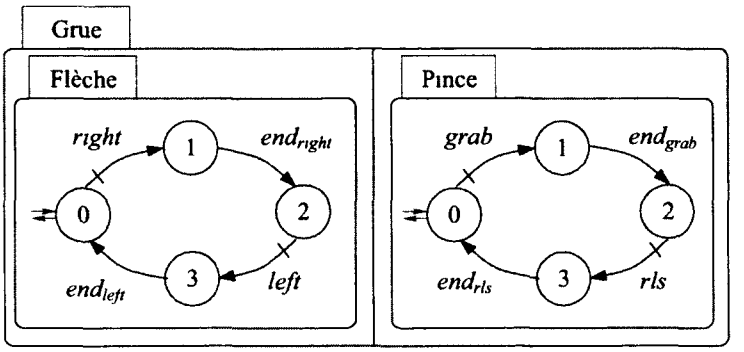
FIGURE 4.2 – Modèle d'un sous-système (problème de contrôle)

pour $q \in Q$.

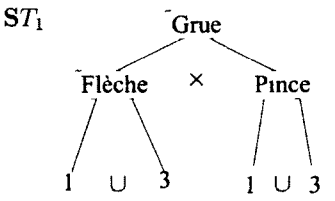
Tel que décrit dans [39] à la section 4.2.1, l'usage de la fonction Θ pour définir P_S résulte en une formule logique définissant une *fonction indicatrice* sur l'ensemble des valeurs que peut prendre la variable $v_{Sous\text{-}Systeme}$. Les termes de base de ces formules logiques ont la forme $v_\alpha = q$ où v_α est une *variable d'état* telle que celles définies dans l'équation 4.3 et q est le symbole dénotant un état que cette variable d'état peu prendre. Par exemple, dans l'équation 4.3, le *domaine* de la variable v_{C_i} est Y_{C_i} pour $i \in \{1, \ldots, n\}$.

Cette situation est illustrée en figure 4.3, qui reprend le problème de la conception d'une grue exposé en section 3.2.2. La grue (le sous-système) est composée d'une flèche et d'une pince en fonctionnement parallèle concurrent. Sa spécification de contrôle est composée de quatre sous-arborescences d'états ST_1 à ST_4. On obtient le prédicat correspondant à la spécification P_S par disjonction des prédicats correspondant à chacune des quatres sous-arborescence d'états de la spécification $\Theta(ST_i)$, $i \in \{1, \ldots, 4\}$. Cette façon de procéder est conforme à ce qui est suggéré dans [39] à la section 4.1. Alors la formule logique $P_S(v_{Sous\text{-}Systeme})$ *exprime* le prédicat de spécification P_S, c'est-à-dire $P_S(v_{Sous\text{-}Systeme}) \equiv P_S$.

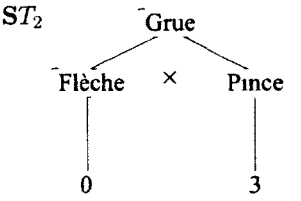
L'introduction des variables d'état de l'équation 4.3 permet ainsi une expression de la spécification directement sous forme d'un ensemble de prédicats, plus compacte que l'énumération d'un ensemble de sous-arborescences d'états. Ces prédicats n'ont cependant pas une forme arbitraire, chacun doit effectivement représenter une sous-arborescence d'états ST_i de la spécification S, donc une version du sous-prédicat $\Theta(ST_i)$. Puisque le STS du



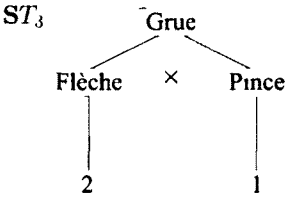
$$\mathcal{S} = \{\mathbf{ST}_1, \mathbf{ST}_2, \mathbf{ST}_3, \mathbf{ST}_4\}$$



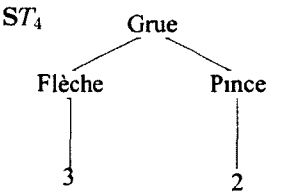
$$P_{\mathcal{S}\Gamma_1} = \Theta(\mathbf{ST}_1) \equiv$$
$$v_{Fleche} \in \{1,3\} \wedge v_{Pince} \in \{1,3\}$$



$$P_{\mathcal{S}\Gamma_2} = \Theta(\mathbf{ST}_2) \equiv$$
$$v_{Fleche} = 0 \wedge v_{Pince} = 3$$



$$P_{\mathcal{S}\Gamma_3} = \Theta(\mathbf{ST}_3) \equiv$$
$$v_{Fleche} = 2 \wedge v_{Pince} = 1$$



$$P_{\mathcal{S}\Gamma_4} = \Theta(\mathbf{ST}_4) \equiv$$
$$v_{Fleche} = 3 \wedge v_{Pince} = 2$$

$$P_{\mathcal{S}} \equiv P_{\mathcal{S}\Gamma_1} \vee P_{\mathcal{S}\Gamma_2} \vee P_{\mathcal{S}\Gamma_3} \vee P_{\mathcal{S}\Gamma_4}$$

FIGURE 4.3 – Spécification de contrôle

sous-système n'a pas de structure verticale, chaque prédicat est une forme conjonctive dont les termes sont des disjonctions exprimant l'ensemble des états actifs d'une composante. Par exemple, dans la figure 4.3, le terme $v_{Flèche} \in \{1,3\}$ de P_{ST_1} correspond à la branche $\text{Grue}^{\text{Flèche}}$ de la sous-arborescence d'états ST_1 et dénote les états actifs de la composante Flèche dans ST_1. Bien sûr, $v_{Flèche} \in \{1,3\} \equiv v_{Flèche} = 1 \vee v_{Flèche} = 3$, il s'agit donc d'une forme du terme de base obtenu par l'application de la fonction Θ. Dans l'expression du prédicat de spécification P_S par une formule logique équivalente $P_S(v_{Sous\text{-}Système})$, il n'est pas nécessaire que l'ordre des termes suive l'ordre des composantes dans la définition du STS (ordre de synthèse) puisque la conjonction logique est commutative. Ceci est un effet de l'absence de structure verticale dans le STS et de l'usage explicite de variables d'état.

Une fois le problème de contrôle spécifié, on le résout par synthèse. L'application des procédures de synthèse de [39] produit une loi de contrôle sous forme d'une SFBC f. Plus spécifiquement ces procédures de synthèse produisent un ensemble de prédicats de synthèse $\{f_\sigma \mid \sigma \in \Sigma_c\}$ *décrivant* une SFBC f (voir la section 2.5) sous forme de BDD (voir la section 2.6 et la section 2.7.3). De cet ensemble de BDD on peut recouvrer un ensemble de formules logiques $\{f_\sigma(v_{Sous\text{-}Système}) \mid \sigma \in \Sigma_c\}$, des fonctions indicatrices sur $v_{Sous\text{-}Système}$, correspondant à l'ensemble des prédicats de synthèse. Une description de ce dernier ensemble constitue une spécification adéquate de la loi de contrôle obtenue. Ainsi pour $\sigma \in \Sigma_c$ arbitraire, $f_\sigma \equiv f_\sigma(v_{Sous\text{-}Système}) \equiv f'_\sigma$ c'est-à-dire que $\{f_\sigma(v_{Sous\text{-}Système}) \mid \sigma \in \Sigma_c\}$ *exprime* adéquatement *deux* lois de contrôle ; l'une applicable à $\textbf{Sous-Système}_{\textbf{STS}}$ (une SFBC f) et l'autre applicable à $\textbf{Sous-Système}_{\textbf{AFD}}$ (une SFBC f') telles que représentées en figure 4.1. On a alors la situation illustrée en figure 4.2, c'est-à-dire que $\textbf{Sous-Système}^f_{\textbf{STS}}$, le sous-système sous le contrôle de f peut être exprimé sous forme d'un AFD soigné

$$\textbf{Sous-Système}^{f'}_{\textbf{AFD}} = (Q^{f'}, \Sigma, \delta^{f'}, q_0, Q^{f'}_m) \,,$$

une description *exhaustive* du sous-système sous le contrôle de f'. L'abus de notation $\textbf{Sous-Système}^f_{\textbf{AFD}} = \textbf{Sous-Système}^{f'}_{\textbf{AFD}}$ est ainsi justifiée, puisque f et f' bénéficient d'une expression commune.

L'élaboration d'un modèle exhaustif de $\textbf{Sous-Système}^f_{\textbf{AFD}}$ n'est en général pas nécessaire pour un sous-système à moins que ce sous-système ne soit la base d'une composante

réutilisable $\mathbf{C} = (\mathbf{G}.\overline{\mathbf{G}})$, car alors

$$\mathbf{G_{ts}} := \textbf{Sous-Système}^{f}_{\textbf{AFD}}$$

forme la structure de transition du modèle de la projection (la machine de Mealy G) tel qu'illustré à la figure 4.4. Cette manoeuvre peut bien sûr être coûteuse car elle équivaut à calculer le prédicat d'accessibilité $R(\textbf{Sous-Système}_{\textbf{STS}}, \neg P_S)$, ce que les procédures de synthèse sur les STS évitent. Mais cette manoeuvre est nécessaire si l'on entend encapsuler le sous-système *Grue* pour former une composante réutilisable car il faut alors formuler le modèle de l'*abstraction* $\overline{\mathbf{G}}$ et donc expliciter G. C'est là une des limitations de cette forme de composantes réutilisables.

Dans un modèle exhaustif tel que celui de la *Grue* illustrée en figure 4.4, la loi de contrôle f est implicite dans $\mathbf{G_{ts}}$ car alors $\Theta(Elig_{\mathbf{G_{ts}}}(\sigma)) = f_\sigma$ pour $\sigma \in \Sigma_c$. Il n'est donc pas nécessaire d'expliciter l'ensemble des formules logiques correspondant aux prédicats de synthèse f_σ si le modèle exhaustif $\mathbf{G_{ts}}$ est suffisamment clair.

Cependant, tel que suggéré dans [39] (section 4.5.1), il est toujours possible d'obtenir un ensemble de prédicats simplifiés en considérant la restriction des prédicats de synthèse f_σ aux configurations où l'événement $\sigma \in \Sigma_c$ est éligible dans le *comportement de la composante qui le génère*. Par exemple dans la figure 4.4 on obtient le prédicat

$$g_{right} := f_{right|_{v_{Flèche}=0}} \equiv v_{Pince} \in \{0, 2\}$$

en limitant le domaine du prédicat f_{right} à $v_{Flèche} = 0$. Alors g_{right}, un cofacteur fonctionnel de f_{right} (voir la section 2.6), est représenté par une formule logique sur la seule variable v_{Pince}. Cette manoeuvre, possible avec des BDD (voir section 4.5 dans [39]), représente une *factorisation* du prédicat original. Le prédicat résultant (ou plutôt sa formule logique) peut alors être considéré comme une *condition de garde* sur la *transition* correspondante de l'AFD *Flèche*. L'*implémentation* de la loi de contrôle f pour le sous-système *Grue* correspond alors à l'imposition de l'ensemble des conditions de garde g_σ de la figure 4.4 sur les *composantes* de la figure 4.3.

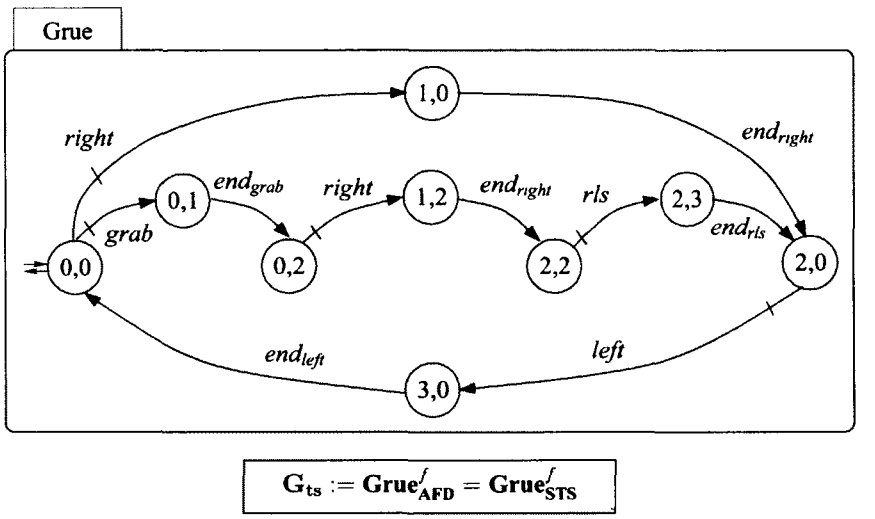


FIGURE 4.4 – Résultat de synthèse pour le sous-système *Grue*

4.1.1.3 Éléments de la spécification d'un sous-système

Les éléments de la spécification d'un sous-système sont donc les suivants :

1. une spécification de la structure du sous-système ;
2. une spécification de contrôle pour le sous-système ;
3. une loi de contrôle résultant de la synthèse.

La discussion qui précède établit que la spécification de contrôle peut être décrite adéquatement par un ensemble de formules logiques $P_S(v_{Sous\text{-}Système})$ décrivant des prédicats de spécification comme à la figure 4.3. Le résultat de synthèse peut alors aussi être donné sous forme d'un ensemble de formules logiques $f_\sigma(v_{Sous\text{-}Système})$ pour $\sigma \in \Sigma_c$ décrivant les prédicats de synthèse correspondant. Les éléments 2 et 3 sont donc adéquatement couverts. Ces deux descriptions reposent cependant sur une définition du *vecteur d'état* du sous-système donné par l'équation 4.3 ; un élément de la *structure* du sous-système. Il faut donc développer une notation adéquate pour décrire la structure du sous-système (1).

La description d'un sous-système doit donner un inventaire détaillé des *ressources* utilisées par le sous-système et en donner la structure. La figure 4.1 donne déjà un aperçu de la structure générale d'un sous-système. Un sous-système est modélisé comme un STS sans structure verticale où l'ensemble des *holons* est assimilé à un ensemble d'AFD générateurs. Il y a bien sûr une légère différence entre les notations utilisées pour décrire des AFD et des *holons*, mais il devrait être clair qu'un AFD décrit adéquatement un *holon* dont la structure externe est vide (voir la section 2.7.1.2). Un diagramme comme celui de la figure 4.1 (A) peut être utile pour décrire la structure du *vecteur d'état* du sous-système et supporter la spécification de contrôle ainsi que la description d'un résultat de synthèse. Mais il lui manque trop de détails concernant l'instanciation de composantes réutilisables et la description des ressources nécessaires pour supporter les éléments de mémoire (entre autres). Il faut aussi considérer que certains sous-systèmes doivent être élaborés de façon *ad hoc* (voir le modèle du vérin à commande monostable de la figure 3.1 à la section 3.1.1.2). Dans ce dernier cas le modèle de la figure 4.1 peut se révéler inapplicable.

Le développement d'une notation appropriée requiert un usage abondant d'illustrations, aussi est-il préférable de différer ce traitement à la section 4.2 qui traite en détail d'exemples de modélisation de composantes réutilisables.

4.1.2 Dynamique d'un sous-système

À ce stade il n'est plus possible de différer le traitement du problème de support à la génération de code.

La spécification d'une composante abstraite, et par extension celle du sous-système qu'elle représente, peut requérir l'adjonction d'heuristiques de génération de code sous forme de *fragments de code*. L'exemple le plus évident est celui de l'abstraction d'un dispositif physique tel que le modèle d'un vérin à commande monostable de la figure 3.1 à la section 3.1.1.2. Il s'agit d'une *composante élémentaire* puisqu'elle est basée sur un sous-système spécifié de façon *ad hoc* à partir du matériel. Sa dynamique repose, entre autres choses, sur l'activité de capteurs qui *génèrent physiquement* des événements incontrôlables devant être explicitement *détectés*. Cette portion de la dynamique du sous-système ainsi que sa loi de contrôle, toutes deux obtenues de façon *ad hoc* dans le cas d'une *composante élémentaire*, doivent être intégrées à la dynamique du système par le générateur de code. Or en général, le générateur de code ne peut pas *déduire* le code nécessaire. Il faut donc que le concepteur *spécifie* ce code dans la structure du sous-système de façon à ce que le générateur de code puisse l'*intégrer* à la dynamique d'exécution du système global selon un schéma prédéfini. Il s'agit de *code d'intégration*, mieux connu par sa terminologie anglophone de *glue code*.

Bien sûr dans le cas d'un sous-système obtenu par assemblage de composantes abstraites (cas des *composantes mixtes*), par exemple la Grue de la figure 4.3, la situation est plus simple. La dynamique du sous-système est alors implicite au modèle de la solution (voir figure 4.4). Le code correspondant peut ainsi être *déduit* par le générateur de code à l'aide du modèle de la solution du problème de contrôle et d'un schéma adéquat de la *dynamique abstraite* d'exécution d'un sous-système. Cependant, même dans ce cas, l'usage de *code d'intégration* peut s'avérer utile sinon nécessaire. Il est donc indispensable de procurer au concepteur une façon de *spécifier* du code d'intégration dans la structure même d'un sous-système.

La spécification d'un sous-système (qu'il soit associé ou non à une composante abstraite) n'est qu'une perspective partielle du système global. Lors de la spécification d'un sous-système le concepteur ne peut donc avoir recours qu'à des *fragments de code* pour

spécifier un *aspect particulier* de la dynamique au sein de ce sous-système, puisque la structure finale de l'application est alors inconnue. De plus, ces *fragments de code* doivent être insérés à des *endroits spécifiques* du code qui implémente la dynamique implicite du sous-système (la solution d'un problème de contrôle : un AFD). Pour pouvoir procurer un tel mécanisme de spécification il faut définir clairement la *dynamique abstraite* d'un sous-système et identifier dans cette dernière un ensemble de *sites* où une intervention est possible.

La *dynamique abstraite* d'un sous-système peut être décrite par un ensemble de *fonctions abstraites* (des algorithmes en pseudo-code) ; un ensemble de *symboles*. Chacun de ces *symboles* constitue alors un *site* d'intervention possible où on peut spécifier :

1. le remplacement intégral de la *définition du symbole* par un fragment de code (définition ou redéfinition du *symbole*) ;
2. l'insertion d'un fragment de code à un endroit précis de la définition du *symbole*.

L'ensemble des *symboles* définissant la *dynamique abstraite* du sous-système constitue alors un ensemble de schémas pour la génération de code procurant la flexibilité nécessaire pour y introduire des *heuristiques* pour la génération du code. Il s'agit bel et bien d'*heuristiques* en ce sens que le code ainsi introduit s'inscrit à l'extérieur du cadre formel de la théorie et peut très bien se révéler inadéquat. Le concepteur est totalement responsable de vérifier ou, du moins, de valider le fonctionnement de ce code.

4.1.2.1 Hypothèses et restrictions

En conception de systèmes par composantes, la conception d'un système se fait par assemblage de pièces détachées, notamment des sous-systèmes résultant de l'instanciation de composantes réutilisables. Ces pièces détachées (les composantes) n'ont qu'une portée limitée au sein de l'application de contrôle et présument en général un certain protocole quant à leur utilisation. La conception de systèmes par composantes présuppose donc une infrastructure prédéterminée procurée par l'architecture de l'application de contrôle. Pour décrire la dynamique abstraite des composantes au sein du système il faut donc poser certaines hypothèses quant à l'architecture de l'application de contrôle.

L'analyse de la section 3.1.2 juge raisonnable la restriction à des applications de contrôle

pour des *usines modulaires*. La section 3.1.3 propose d'envisager l'implémentation de ces applications de contrôle sur un PLC (un automate programmable). Ce genre d'élément de contrôle (ici un PLC) utilise une rétroaction cyclique selon le schéma général en trois phases d'écrit en section 3.1.3. Le *calcul* de la rétroaction (c'est-à-dire l'évaluation de la commande) constitue l'*une* des phases du cycle. La fonction de rétroaction est donc évaluée systématiquement à chaque cycle de la rétroaction. Au sens de Sifakis [25] il s'agit d'une architecture (ou d'un modèle d'exécution) d'application *synchrone*.

Dans un modèle d'exécution synchrone l'application de contrôle fonctionne par cycle ; il s'agit d'un répartiteur cyclique (de l'anglais *cyclic executive*, voir [49]) souvent initié à interval fixe. Dans le cas des usines modulaires, un nouveau cycle est initié sitôt le précédent cycle terminé. À chaque cycle tous les éléments du système sont *activés* dans un ordre prédéterminé et se voient accorder une *tranche de temps* pour leurs calculs.

On peut donc présumer que l'application de contrôle *générée* pour le genre de composantes préconisées ici *active* périodiquement chaque sous-système (instance de composantes réutilisables) par un mécanisme quelconque (par exemple un appel de procédure). La fréquence d'appel peut varier, mais on présume que le temps de calcul requis pour la rétroaction est beaucoup plus court que le temps de réaction moyen des dispositifs de l'usine modulaire. À chaque cycle, avant le calcul de la rétroaction, l'application met à jour une image mémoire de l'*état de l'usine* (capteurs, actionneurs et registres d'entrée de périphériques) dont les sous-systèmes peuvent ensuite se servir pour leurs calculs.

4.1.2.2 Description de la dynamique abstraite

Pour plus de généralité il est préférable de décrire la dynamique abstraite des sous-système dans le contexte de composantes réutilisables, c'est-à-dire lorsqu'ils sont associés à une *projection* (l'interface de la composante réutilisable). Le modèle de la dynamique abstraite d'un sous-système sans projection peut ensuite être obtenu par simplification de la dynamique abstraite générale.

On présume que l'*environnement d'exécution* d'un sous-système contient implicitement quatre fonctions abstraites pour chaque événement $\sigma \in \Sigma$: $func_\sigma()$, $P_\sigma()$, $Project_\sigma()$ et $lal_\sigma()$. L'*environnement d'exécution* d'un sous-système contient aussi une fonction abstraite pour chaque événement $\tau \in T$ de sa projection associée : $func_\tau()$.

L'algorithme 4.1 illustre le schéma de définition pour la fonction $func_\sigma()$; elle retourne un résultat booléen. La fonction $P_\sigma()$ correspond au prédicat de synthèse et retourne un résultat booléen. L'algorithme 4.2 illustre le schéma de définition pour la fonction $Project_\sigma()$ qui retourne aussi un résultat booléen. La fonction $lal_\sigma()$ est en fait une procédure et ne retourne aucun résultat. La notation utilisée ici est assez libérale et permet entre autres d'appeler une fonction sans en utiliser le résultat dans une expression. La fonction $func_\tau()$ retourne un résultat booléen. L'ensemble de ces fonctions définissent complètement la dynamique abstraite. Une brève explication s'impose.

Premièrement, il est présumé qu'à l'exécution, la génération d'un événement $\sigma \in \Sigma$ dans un sous-système provoque l'exécution de la fonction $func_\sigma()$ qui lui correspond. Alors $func_\sigma()$ vérifie si σ est *éligible* dans le sous-système local étant donné son *état courant* (appel à $P_\sigma()$). Si σ n'est pas éligible, le traitement s'arrête et $func_\sigma()$ retourne le résultat *false*, provoquant l'arrêt de l'activité liée à cet événement. Si au contraire σ est éligible, alors σ doit d'abord être propagé au *client* de la composante à travers son *interface* (appel à $Project_\sigma()$).

Il est important de noter que $P_\sigma() := f_\sigma(v_{Sous\text{-}Système})$, le prédicat de synthèse, et que dans la description du comportement contrôlé du sous-système ($\mathbf{G_{ts}}$) σ n'est éligible que si et seulement si $f_\sigma(v_{Sous\text{-}Système})$ est vérifié pour l'état courant du sous-système. Ainsi, si σ est *incontrôlable* ($\sigma \in \Sigma_u$), σ est *toujours* éligible dans le sous-système puisque $\mathbf{G_{ts}}$ exprime un résultat de synthèse.

Deuxièmement, $Project_\sigma()$ implémente la dynamique de la projection qui est définie par la *machine de Mealy* G associée à la composante. Elle vérifie si oui ou non une

Algorithme 4.1: $func_\sigma()$

```
1 if P_σ() then /* Elig_{G_ts}(σ) : σ éligible dans le sous-système local ? */
2     if Project_σ() then /* σ permis par le système global ? */
3         lal_σ() /* Exécuter la liste des actions locales pour une occurrence de σ ; */
4         return true
5     end
6 end
7 return false
```

Algorithme 4.2: $Project_{\sigma}()$

```
/* q ∈ Q désigne l'état courant du sous-système */
1 if (∃(q, σ) ∈ Q × Σ) : λ(q, σ) = τ ∈ T then
2     return func_τ()
3 else
4     return true
5 end
```

transition sur σ effectuée à partir de l'état courant du sous-système local est projetée. Lorsque l'événement n'est pas projeté, c'est-à-dire $\lambda(q, \sigma) = \epsilon$ ($q \in Q$ dénotant l'état courant du sous-système local), $Project_{\sigma}()$ retourne simplement le résultat *true*. Autrement ($\lambda(q, \sigma) = \tau \in T$ dans G), $Project_{\sigma}()$ retourne le résultat de l'invocation de la fonction $func_{\tau}()$.

Il est important de noter que les fonctions $func_{\tau}()$ ($\tau \in T$) correspondant à l'interface d'une composante réutilisable ne sont définies *qu'à l'instanciation* de cette composante dans un autre sous-système : le *sous-système client*. Le concept est similaire à celui de *méthode abstraite* d'un langage de classe à la différence que l'*instanciation* (et non l'*héritage*) procure le mécanisme et le *site* de la définition. À la conception d'une composante réutilisable les fonctions $func_{\tau}()$ sont déclarées implicitement dans l'environnement local mais elles n'y sont *pas définies*. Le mécanisme d'instanciation est responsable de leur assigner ultérieurement une définition dans un *sous-système client*. Cette définition prend la forme d'une association (liaison statique puisqu'il n'y a pas d'héritage) avec une fonction $func_{\sigma}()$ définie dans l'environnement d'un *sous-système client*. De cette façon on procure un lien (d'appel de fonction) à partir de l'environnement local d'une instance de composante réutilisable vers l'environnement de son *sous-système client*. Tout appel d'une fonction $func_{\tau}()$ dans l'environnement local d'une composante réutilisable est implicitement *redirigé* vers une fonction $func_{\sigma}()$ du sous-système qui instancie cette composante.

Finalement, lorsque $Project_{\sigma}()$ retourne le résultat *true*, la procédure $lal_{\sigma}()$ est appelée. Cette procédure implémente la *liste des actions locales* requises pour *mettre à jour l'état courant* du sous-système local. Pour un sous-système comme celui illustré en figure 4.1 ceci comprend la mise à jour des variables du vecteur d'état (équation 4.3).

Il devrait être clair que dans la dynamique d'un *sous-système sans projection*, la fonction $Project_\sigma()$ se comporte comme une tautologie. On peut donc l'éliminer et corriger $func_\sigma()$ en conséquence c'est-à-dire éliminer la conditionnelle sur $Project_\sigma()$ complètement.

Similairement, lorsque $\sigma \in \Sigma_u$, le prédicat $P_\sigma()$ est une tautologie. On ne peut cependant pas éliminer l'appel à $Project_\sigma()$ dans $func_\sigma()$ puisqu'il faut propager l'occurrence de l'événement dans l'abstraction, s'il y a projection, de sorte à mettre à jour l'état du système globalement. Mais puisque tout événement projeté est alors sûrement incontrôlable, à cause des règles suivies pour l'abstraction de composantes (voir section 3.4.2.1), l'appel à $Project_\sigma()$ ne peut retourner que le résultat *true* et la conditionnelle devient inutile. La fonction $func_\sigma()$ prend ainsi la forme simplifiée de l'algorithme 4.3.

On dispose donc maintenant de l'équipement nécessaire pour introduire des heuristiques de génération de code dans la spécification de sous-systèmes (et donc de composantes réutilisables), notamment l'ensemble des *symboles* : $func_\sigma()$, $P_\sigma()$, $Project_\sigma()$ et $lal_\sigma()$ définis implicitement pour chaque événement du sous-système. Chacun de ces symboles peut être soit *redéfini intégralement* ou par *insertion d'un fragment de code* à l'aide d'une notation appropriée.

Finalement, pour *lier* ces heuristiques à l'architecture de l'application de contrôle, on introduit une dernière fonction abstraite, *cycle*(), initialement vide, que le concepteur peut utiliser pour activer les autres éléments de la dynamique abstraite (les autres fonctions abstraites). En particulier, pour les composantes élémentaires, le concepteur est responsable d'introduire dans cette fonction le code nécessaire pour activer les fonctions $func_\sigma()$, puisque le sous-système est alors spécifié de façon *ad hoc*.

Algorithme 4.3: $func_\sigma()$ sur événements incontrôlables

```
1 Project_σ() /* projeter l'événement s'il y a lieu ; */
2 lal_σ() /* Exécuter la liste des actions locales pour une occurrence de σ. */
3 return true
```

4.2 Modélisation de composantes

On peut maintenant procéder au développement d'une notation convenable pour éventuellement spécifier une bibliothèque de composantes réutilisables et illustrer notre étude de cas. La notation développée dans ce qui suit est adaptée aux besoins de la présente thèse et vise à réduire l'encombrement des figures. Les limitations imposées ici n'entraînent pas de perte de généralité. On peut cependant faire beaucoup mieux si l'on vise l'implémentation d'une bonne interface usager graphique par exemple.

Une composante réutilisable $\mathbf{C} = (\mathbf{G}, \overline{\mathbf{G}})$ telle que proposée s'articule nécessairement autour du modèle exhaustif de la solution à un problème de contrôle (le sous-système $\mathbf{G_{ts}}$), sur lequel on greffe le modèle d'une interface ; $\overline{\mathbf{G}}$. La spécification d'une composante réutilisable contient donc cinq éléments ; la description du problème de contrôle :

1. une spécification de la structure d'un sous-système ;
2. une spécification de contrôle pour le sous-système ;
3. une loi de contrôle résultant de la synthèse ;

et une description de la projection (l'interface) :

4. une représentation de $\mathbf{G}$;
5. une représentation de $\overline{\mathbf{G}}$.

On distingue deux types de composantes réutilisables :

1. les composantes dites *élémentaires*, qui constituent une abstraction initiale d'un dispositif physique, et dont le problème de contrôle doit être spécifié et résolu de façon *ad hoc* ;
2. les composantes dites *mixtes* basées sur des problèmes de contrôle formulés par assemblage d'instances d'autres composantes réutilisables et résolus de façon formelle avec les procédures de SCT.

4.2.1 Règles et conventions

Lors de l'élaboration du modèle d'une composante, les ressources physiques telles que capteurs, actionneurs et éléments de mémoire ne sont pas *assignées* a priori. La définition de ces éléments dans l'*environnement local* n'est donc pas finale et joue le même rôle

qu'une déclaration de variables dans un langage de programmation. Une notation similaire à celle d'un langage de programmation peut être utilisée au détriment de l'espace requis dans les figures. Puisque cette étude vise la production de code pour des automates programmables (PLC), que le nombre de types d'éléments dans cet environnement est limité et que l'illustration au moyen de figures est une nécessité, l'usage d'une notation par préfixe est, ici, préférable à une notation plus extensive.

4.2.1.1 Variables et type de données

Pour l'usage de types de données dans la spécification d'une composante, certaines conventions sont utiles. L'ensemble des préfixes suivant dénote des types de données :

1. le préfixe «X» dénote un capteur binaire ;
2. le préfixe «Y» dénote un actionneur binaire ;
3. le préfixe «M» dénote un élément de mémoire booléen ;
4. les préfixes «XW» et «XD» dénotent des registres d'entrée de périphériques de types respectifs entier (en complément à deux) et réel ;
5. les préfixes «YW» et «YD» dénotent des registres de sortie de périphériques de types respectifs entier (en complément à deux) et réel ;
6. les préfixes «MB», «MW» et «MD» dénotent des registres de mémoire interne de types respectifs octet, entier et réel ;
7. le préfixe «T» dénote une minuterie ;
8. le préfixe «SR» dénote un registre à décalage.

En général, pour désigner des variables de ces types, on utilise simplement des numéros d'éléments préfixés sans recourir à une déclaration préalable, par exemple :

- des capteurs : X, X0, X1 ;
- des actionneurs : Y, Y0, Y1 ;
- le registre de lecture associé à un micromètre : XW2 ;

et ainsi de suite. L'usage de *variables typées* n'est pas exclu, entre autres pour rendre un modèle plus *expressif*, pourvu que ces variables fassent l'objet d'une déclaration préalable dans une section appropriée, par exemple :

```
VAR
  VLD, OK : M; /* indicateurs booléens */
  Mem1 : 0..5; /* variable d'état
                  d'un élément de mémoire */
ENDVAR
```

À ce titre l'usage de types énumérés (implicitement de type entier), comme pour la variable `Mem1` précédemment, procure plus de clarté.

Par convention, les événements de fronts montant et descendant sur des types booléens sont prédéfinis par exemple, pour un capteur X0 :

- $\uparrow$X0 désigne le front montant ;
- $\downarrow$X0 désigne le front descendant.

La concaténation d'événements peut aussi être déclarée, par exemple `right = ↓Y1·↑Y0` définit une *macro commande* à partir de deux événements contrôlables ; on peut aussi simplement l'utiliser explicitement dans un modèle. De la même façon diverses constantes peuvent être déclarées, par exemple $F_0 \equiv \text{X0} \wedge \text{Y1}$ pour une formule logique fréquemment utilisée.

4.2.1.2 Signatures d'instanciation

Certaines ressources telles que les capteurs, les actionneurs, les registres d'entrée et sortie de périphériques ainsi que les minuteries sont représentées par des *variables*. On a par exemple Y0 pour un capteur booléen et T0 pour une minuterie. Avant la génération de code, ces *variables* doivent être assignées à des ressources réelles de même type. Aussi, certaines ressources requièrent des paramètres pour leur fonctionnement ; par exemple une minuterie requiert éventuellement un *délai* d'expiration. Ces éléments ne sont généralement pas requis avant la phase finale de génération d'un système et il est encombrant d'avoir à les spécifier *a priori* au moment de l'instanciation de composantes ou, pire encore, d'avoir à les définir comme constantes dans un modèle. Pour procurer un mécanisme flexible permettant au concepteur de spécifier ces paramètres *a posteriori*, la spécification de composantes réutilisables admet la définition d'une *signature d'instanciation*. Par exemple la signature `Jack111<Y0,X0,X1>` signifie que la composante réutilisable `Jack111` utilise un ac-

tionneur et deux capteurs. Cette signature agit comme une section de déclaration. Dans le corps de la définition de `Jack111` les variables Y0, X0 et X1 sont déjà définies.

La signature d'instanciation n'est pas limitée aux types prédéfinis. Ainsi la signature `Micrometer<XW0, const loLimit, hiLimit : MW>` spécifie une *plage de valeurs* (`loLimit` et `hiLimit`, des constantes) pour la validité d'une lecture sur le registre d'entrée de périphérique XW0. La particularité de la signature d'instanciation est d'accepter une *spécification partielle à l'instantiation*. Ainsi une déclaration d'instance telle que `Boom : Jack111` au sein d'une définition de sous-système définit la variable `Boom` sans assigner aucune de ses ressources. La déclaration `Boom : Jack111<Y0 := Q3.4>`, aussi acceptable, assigne l'actionneur et diffère l'assignation des capteurs sans impact sur la phase de modélisation.

La génération de code assigne la mémoire interne plus efficacement qu'une spécification explicite, par conséquent les éléments de mémoire n'apparaissent généralement pas dans une signature d'instanciation. Une déclaration comme `const loLimit : MW` dans une signature d'instanciation ne définit que le *type* de la constante.

4.2.1.3 Types de données structurés

Deux types prédéfinis vus précédemment sont inusités : les minuteries (préfixe «T») et les registres à décalage (préfixe «SR»). Ces deux type peuvent être considérés comme des éléments de mémoire d'un PLC. Mais alors qu'une minuterie *génère* un événement (l'élément de mémoire est donc *actif*), un registre à décalage n'a pour sa part qu'une structure et un comportement particulier. Les deux sont liés à une dynamique prédéterminée procurée par des primitives du PLC et méritent une explication.

Les registres à décalage définissent des chaînes de bits sur lesquelles un PLC peut appliquer diverses primitives de *décalage*, donnant à l'ensemble un comportement stéréotypé connu dans la plupart des langages de programmation. Il n'est pas possible cependant de donner une représentation d'un registre à décalage par un AFD unique couvrant toutes ses possibilités. Il n'est pas possible non plus de lui attribuer un événement qui lui appartienne en *propre*, c'est-à-dire que *génère* le registre à décalage, car en général l'opération de décalage doit être *synchronisée* sur l'activité d'autres dispositifs. Par convention, dans notre contexte, on utilise les registres à décalage comme tous les autres éléments de mémoire,

c'est-à-dire comme l'expression explicite de contraintes pour le contrôle. La seule différence est que l'opération de décalage est utilisée pour modéliser certaines transitions et dispose d'un support au niveau du PLC.

La figure 4.5 donne un exemple de modèle mémoire utilisant un registre à décalage. Puisqu'il s'agit d'un modèle mémoire, l'événement *shift* est partagé dans l'environnement local. On peut voir dans la figure que l'AFD n'est pas *fermé* sur l'événement *shift* de sorte que si cet événement est incontrôlable, il faut inclure dans la spécification de contrôle des *contraintes* assurant le comportement décrit par l'AFD.

Par convention, la déclaration d'un registre à décalage est toujours assortie d'une description de sa structure. Le registre à décalage de la figure 4.5 correspond à une déclaration comme `Tbl : SR<M0,M1,M2,M3>` si on présume que l'élément de mémoire porte le nom symbolique `Tbl`. Cette façon de faire procure une structure pour le *vecteur d'état* de l'AFD et permet l'expression de chaque état par une configuration de la chaîne de bits comme dans la figure. Elle permet aussi l'expression de formules logiques plus simples pour exprimer des espaces d'états de l'AFD. Par exemple, la formule $\overline{\mathrm{M0}}$ remplace avantageusement une formule plus complexe comme $v_{Tbl} \in \{0, 1, 2, 3, 4, 5, 6, 7\}$.

Les minuteries sont des dispositifs internes (dans les PLC) faisant usage d'éléments de mémoire : un *compteur* et un *indicateur d'échéance* de type booléen. Il existe une grande variété de définitions de ces dispositifs implémentés par diverses primitives. Par convention on peut se limiter à une seule forme ; la minuterie de *délai à la montée* illustrée de façon schématique par l'AFD de la figure 4.6 (a).

On peut accéder de façon symbolique à l'*indicateur d'échéance* en utilisant le même symbole que celui désignant la minuterie (par exemple T dans la figure). La formule logique T est donc vérifiée lorsque la minuterie est *échue*, autrement la formule logique $\overline{\mathrm{T}}$ est vérifiée. Aussi une minuterie *génère*-t-elle deux événements, $\uparrow$T et $\downarrow$T, considérés *incontrôlables*. Cependant la minuterie dépend d'un *signal* (dénoté S dans la figure 4.6 (a)) pour son *activation*. Ce *signal d'activation* (une formule logique) réfère normalement à l'*état* d'éléments situés dans l'environnement local (mémoires, capteurs ou autres). Ainsi une minuterie *partage* généralement l'*état* d'autres dispositifs du sous-système (comme illustré à la figure 4.6 (b)) et en *enrichit le comportement*. Cette caractéristique confère aux minuteries une grande flexibilité, mais elle en limite aussi les possibilités d'abstraction.

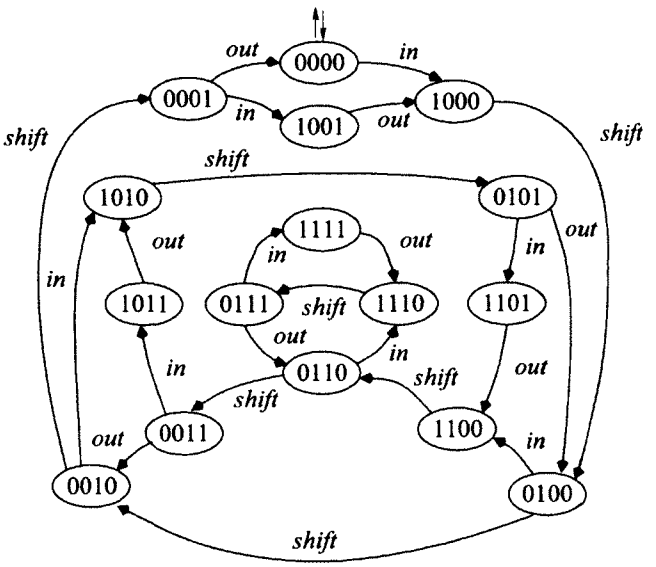
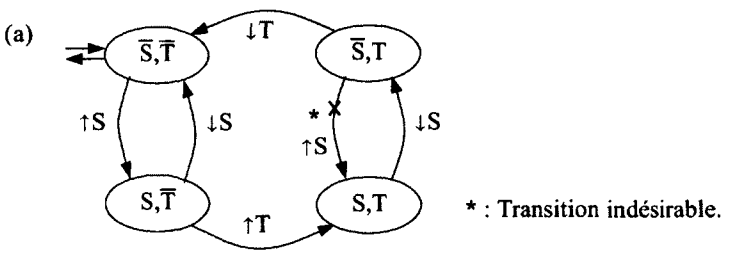


FIGURE 4.5 – Mémoire utilisant un registre à décalage



(b) $S := Y0 \wedge \overline{X0}$

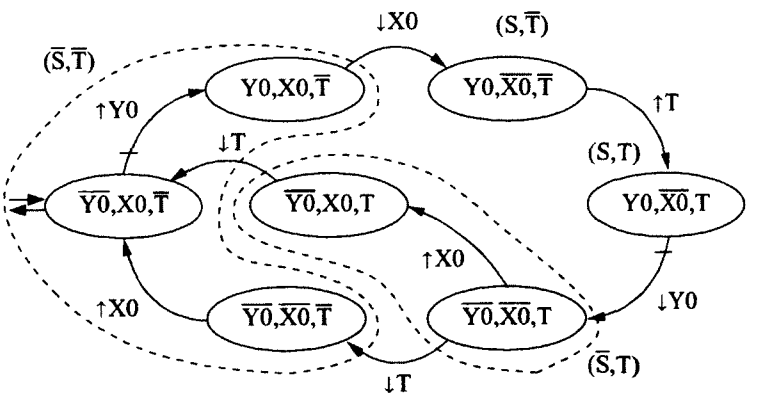


FIGURE 4.6 – Minuterie de délai à la montée (schéma et application)

La figure 4.6 (a) illustre le comportement du *dispositif brut* d'une minuterie de délai à la montée. Initialement la minuterie est *inactive* : état $\langle \overline{S}, \overline{T} \rangle$. La minuterie est *évaluée* à chaque cycle de rétroaction selon son *signal d'activation* S. Lorsque S est réalisé la minuterie devient *active* : état $\langle S, \overline{T} \rangle$. La minuterie reste *active* tant que le signal d'activation S est réalisé et que son *délai d'échéance* n'est pas écoulé. Elle est réarmée et redevient *inactive* sitôt que $\overline{S}$ est réalisé. Si S demeure réalisé pour un interval de temps supérieur ou égal à au *délai d'échéance* spécifié, la minuterie passe à l'état *échue* : $\langle S, T \rangle$. Une fois la minuterie *échue* on ne peut que la réarmer (événement $\downarrow$S). Il faut bien sûr que $\overline{S}$ persiste jusqu'à la prochaine évaluation de la minuterie (un cycle de rétroaction) pour qu'elle soit complètement réarmée et repasse à l'état *inactive*. Cette condition doit être *vérifiée* manuellement à la conception (*ad hoc*) du sous-système (par exemple figure 4.6 (b)), d'où la transition indésirable dénotée par une *astérisque* dans la figure 4.6 (a).

Les *dispositifs bruts* de minuterie procurés par les PLC rendent impossible de spécifier l'état *échue* d'une minuterie de délai à la montée comme état initial. Pour plus de flexibilité on utilise des minuteries mémorisées. Elles sont implémentées *par programmation* à partir du dispositif brut d'une minuterie de délai à la montée pour produire deux types de comportement :

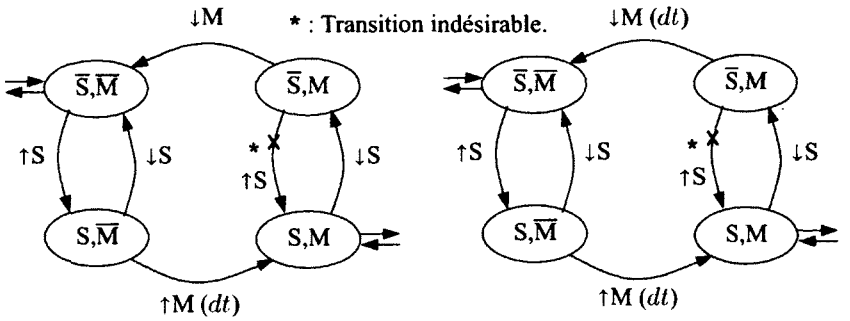
1. ODT (*On Delay Timer*) délai *à la montée seulement* illustré en figure 4.7 (a) ;
2. DDT (*Dual Duty Timer*) délai *à la montée et à la descente* illustré en figure 4.7 (b).

On substitue un élément de mémoire M à l'indicateur d'échéance T pour pouvoir démarrer la *minuterie brute* dans n'importe laquelle des deux configurations $\langle \overline{S}, \overline{M} \rangle$ ou $\langle S, M \rangle$. Les minuteries mémorisées requièrent quatre *paramètres* illustrés en figure 4.7 (a) et (b) de façon schématique :

- un type de *comportement* (dénoté ODT ou DDT) ;
- une *formule logique* définissant le *signal d'activation* (dénoté S) ;
- un *indicateur d'échéance* (une formule logique de la forme M ou $\overline{M}$ où M désigne un élément de mémoire booléen) ;
- un délai de minutage (dénoté *dt*).

Une minuterie mémorisée doit être déclarée explicitement dans la structure d'un sous-système pour permettre d'y associer ces paramètres. Par exemple la déclaration suivante

```
T0 : ODT[S := Y0 ∧ X1, M0, dt];
```



(a) T : ODT[S, M | $\overline{M}$, *dt*] (b) T : DDT[S, M | $\overline{M}$, *dt*]

FIGURE 4.7 – Minuteries mémorisées

définit une minuterie mémorisée de *délai à la montée* du signal S (type ODT), défini par la formule logique $\overline{Y0} \wedge \overline{X1}$, au bout d'un délai exprimé par une constante *dt*. L'indicateur d'échéance M0, exprimé en *forme positive*, déclare implicitement l'élément de mémoire booléen M0 ainsi que l'état initial de la minuterie : la minuterie est *échue* à l'état initial ($\langle S, M \rangle$ dans la figure 4.7 (a)). Par opposition, la déclaration

```
T0 : ODT[S := Y0 ∧ X1, M0, dt];
```

définit la même minuterie avec les mêmes paramètres mais dont l'état initial, dénoté par la formule $\overline{M0}$ en *forme négative*, est *inactive* ($\langle \overline{S}, \overline{M} \rangle$ dans la figure 4.7 (a)). Ces déclarations peuvent être utilisées à l'intérieur d'une déclaration de composante, par exemple

```
Define component
  Jack101<Y0, T0, X1, const dt : TIME> ...
```

où `TIME` est un type particulier pour les constantes de temps et T0 représente l'identificateur auquel on doit éventuellement *assigner* un dispositif de *minuterie brute* pour la génération de code.

Finalement, pour implémenter la dynamique de la minuterie, le concepteur doit fournir un fragment de code. La forme générale pour le comportement ODT est la suivante :

```
/* retour à l'état initial. */
if (( Y0 ∨ X1 ) ∧ M0) then func↓M0();
/* activités à l'échéance */
if ( T0 ∧ Y0 ∧ X1 ∧ M0 ) then func↑M0();
```

```
/* évaluation de la minuterie */
eval_T(T0, Y0 ∧ X1 ∧ M0);
```

en supposant la déclaration précédente de T0. Ce code doit être activé *à chaque cycle* de la rétroaction et doit donc être introduit dans l'heuristique de génération de code *cycle*() défini à la fin de la section 4.1.2.2. L'activité variable sur occurrence des événements ↑M0 et ↓M0 est supposée décrite dans la dynamique locale ($lal_{\uparrow M0}()$ et $lal_{\downarrow M0}()$). Dans ce qui suit il revient au concepteur d'introduire le code correspondant aux endroits appropriés dans la spécification.

4.2.2 Composantes élémentaires

Les composantes élémentaires représentent des abstractions effectuées à partir du modèle exhaustif *ad hoc* d'un dispositif matériel comme présenté à la section 3.1.1.2. Par opposition, le sous-système d'une composante *mixte* est constitué d'instances d'autres composantes réutilisables (mixtes ou élémentaires) en fonctionnement parallèle concurrent. Pour illustrer quels sont les éléments d'une spécification de composante réutilisable élémentaire, l'exemple simple du vérin à commande monostable de la section 3.1.1.2 suffit. Pour des modèles plus complexes impliquant des minuteries et des éléments de mémoire, le lecteur peut se référer au catalogue de composantes de l'annexe D contenant l'ensemble des modèles nécessaires à l'élaboration de *l'étude de cas* du chapitre 5.

La figure 4.8 élabore la structure du sous-système correspondant à un vérin à commande monostable doté d'un ensemble complet de capteurs. En conjonction avec la figure 4.9 (le modèle de la projection) et de la figure 4.10 (le modèle d'interface), ces éléments constituent une spécification complète de la *composante élémentaire* `Jack111`. Le sous-système correspondant à une composante élémentaire (`Basic Component`) doit être obtenu de façon *ad hoc*. La structure du vecteur d'état (énoncé `State Vector` dans la figure 4.8) doit donc être donnée *a priori*. On peut utiliser le vecteur d'état pour obtenir des *signatures logiques* pour chaque état. Par exemple, en figure 4.9, l'état initial dénoté <$\overline{Y}$, X0, $\overline{X1}$> correspond à la formule logique $\overline{Y} \wedge X0 \wedge \overline{X1}$. Ces *signatures logiques* sont nécessaires pour établir la *fonction de rétroaction* (une SFBC) qui doit aussi être donnée *a priori* dans le cas des sous-systèmes élémentaires. Dans la figure 4.8 la fonction de rétroaction est donnée de

façon exhaustive par les deux prédicats $f_{\uparrow Y}$ et $f_{\downarrow Y}$, puisque $\uparrow$Y et $\downarrow$Y sont les seuls événements contrôlables (liés au *capteur* Y). On se rappelle que dans une SFBC les prédicats liés aux événements incontrôlables sont des tautologies. Finalement, puisque dans un sous-système la définition par défaut des fonctions $P_\sigma()$ correspond au prédicat de rétroaction, les fonctions $f_{\uparrow Y}$ et $f_{\downarrow Y}$ définissent implicitement $P_{\uparrow Y}()$ et $P_{\downarrow Y}()$.

Dans la figure 4.8, seuls les éléments *modifiés* de la dynamique abstraite (les heuristiques de génération de code) sont explicitement décrits. Tous les éléments conformes à la description de la section 4.1.2.2 n'ont pas à être explicités puisqu'ils peuvent être générés tels quels. Ainsi les fonctions $func_{\uparrow Y}()$ et $func_{\downarrow Y}()$ sont conformes à l'algorithme 4.1 (événements contrôlables) et les fonctions $func_{\uparrow X1}()$, $func_{\uparrow X0}()$, $func_{\downarrow X1}()$ et $func_{\downarrow X0}()$ sont conformes à l'algorithme 4.3 (événements incontrôlables).

Dans le sous-système d'une composante élémentaire, les functions $Project_\sigma()$ sont considérées vides (elle retournent toutes la valeur *true*). Puisque le système est conçu de façon *ad hoc* c'est au concepteur qu'il revient d'en changer la définition. Les fonctions $Project_\sigma()$, pour $\sigma \in \{\uparrow Y, \uparrow X1, \downarrow Y, \uparrow X0\}$, sont ainsi redéfinies puisque ce sont les seuls événements associés à une projection dans la figure 4.9. Les événements $\downarrow$X0 et $\downarrow$X1 sont *effacés* par la projection et ainsi la définition par défaut de $Project_\sigma()$ leur est adéquate. La fonction $func_{deploy}()$ projetant l'événement *deploy* à partir de l'événement $\uparrow$Y n'est ici qu'un *symbole abstrait* sans définition. Ce symbole doit être associé à une définition par l'*utilisateur* de la composante réutilisable `Jack111` au moment de l'instanciation de cette dernière dans un sous-système.

D'une façon similaire, dans le sous-système d'une composante élémentaire, les functions $lal_\sigma()$ sont considérées vides *a priori*. Pour $\sigma \in \{\uparrow X0, \downarrow X1, \uparrow X0, \downarrow X1\}$, cette définition convient, puisque l'occurrence de ces événements ne modifie pas l'*environnement local*. Cependant $lal_{\uparrow Y}()$ et $lal_{\downarrow Y}()$ modifient l'état de l'actionneur du vérin et doivent être redéfinies en conséquence.

À chaque itération de la boucle de rétroaction le système doit exécuter le code d'intégration (*glue code*) des composantes. Ce code est généralement inséré manuellement dans l'heuristique de génération de code *cycle*(). Pour lier l'activité des dispositifs physiques à la dynamique abstraite il faut insérer du code correspondant dans *cycle*(). Sur occurrence d'un événement incontrôlable *cycle*() doit *détecter* cette occurrence et activer la fonction

```
Basic Component Jack111<Y, X0, X1> {

  SubSystem Jack111 {
    State Vector = <Y, X0, X1>

    f↑Y := ¬Y ∧ X0
    f↓Y := Y ∧ X1

    Project↑Y() := { return func_deploy(); }
    Project↑X1() := { return func_deployend(); }
    Project↓Y() := { return func_retract(); }
    Project↑X0() := { return func_retractend(); }

    lal↑Y() := { set(Y); }
    lal↓Y() := { rst(Y); }

    cycle() := cycle() + {
      if ( ↑X1 ) then func↑X1();
      if ( ↑X0 ) then func↑X0();
      func↑Y();
      func↓Y();
    }
  }
}
```

FIGURE 4.8 – Éléments de structure du sous-système Jack111

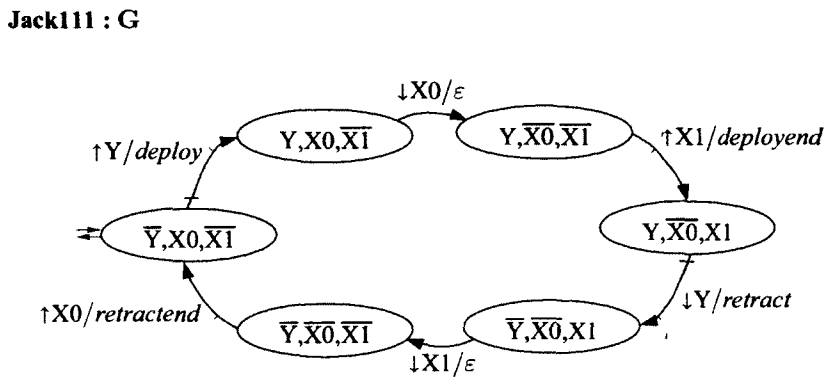


FIGURE 4.9 – Composante Jack111, sous-système et projection

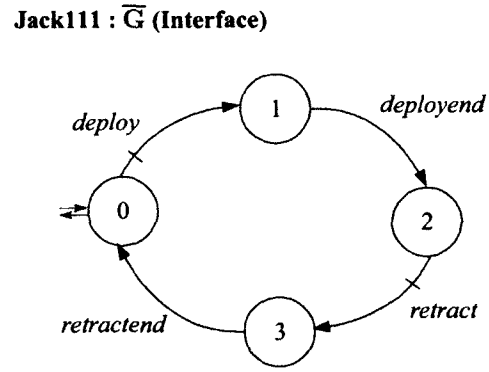


FIGURE 4 10 – Composante Jack111, interface

$func_\sigma()$ qui lui correspond. Quant aux événements contrôlables, puisqu'ils sont générés par la boucle de rétroaction elle-même, *cycle*() doit toujours invoquer la fonction $func_\sigma()$ correspondant à un événement contrôlable pour lui permettre de se produire lorsqu'il est éligible localement et permis par l'environnement du système global.

Il est intéressant de noter qu'aucun des événements ↓X1 et ↓X0 ne sont effacés par la projection et qu'ils ne causent aucun changement à l'environnement local du sous-système de la composante `Jack111`. Ainsi l'invocation de $func_{\downarrow X1}()$ et $func_{\downarrow X0}()$ ne produit aucune activité utile dans le système et peut donc être éliminée.

Par convention la redéfinition d'un symbole de la dynamique abstraite est signalée par une expression utilisant l'opérateur d'affectation suivi d'une définition, comme par exemple la redéfinition de la fonction $lal_{\uparrow Y}()$ dans la figure 4.8. L'insertion de code est signalée par une forme similaire à celle utilisée pour *cycle*() dans la même figure. La forme $cycle() := cycle() + \{...\}$ signale une insertion à la fin de la procédure et la forme $cycle() := \{...\} + cycle()$ une insertion au début. Aussi, seules les heuristiques *cycle*() et $lal_\sigma()$ admettent l'insertion de code. Les autres n'admettent que la redéfinition.

Finalement la figure 4.10 spécifie l'interface de la composante réutilisable sous forme d'un AFD quotient $\overline{G}$ de l'AFD G de la figure 4.9. Le quotient $\overline{G}$ constitue un *observateur* de G. Il est obtenu par application de l'algorithme de Wong et Wonham dans [60]. On peut en général assigner aux états de l'interface des étiquettes pour rendre le modèle plus expressif. Ces étiquettes tiennent généralement lieu d'indices que l'on peut affecter à une *variable d'état* comme celles définies dans l'équation 4.3 pour désigner l'*état courant* d'une instance de composante d'un sous-système. Pour réduire l'encombrement des figures on n'utilise ici que de simples index explicites.

4.2.3 Composantes mixtes

Les composantes mixtes sont conçues à partir de sous-systèmes qui eux-mêmes sont assemblés par instanciation de composantes réutilisables. Cette façon de procéder permet un processus de conception basé sur l'usage de la synthèse SCT pour spécifier un *contrôle local* sur le sous-système. La spécification *ad hoc* du contrôle (typique des composantes élémentaires) est ainsi remplacée par une spécification formelle des requis de contrôle et

une loi de contrôle obtenue par synthèse. Les modèles $\mathbf{G}$ et $\overline{\mathbf{G}}$ (de la projection et de l'interface respectivement) peuvent être donnés de la même façon que pour les composantes élémentaires. La structure du sous-système de la composante mixte doit cependant détailler l'instanciation de composantes réutilisables ainsi que leur assemblage et fournir une spécification de contrôle. Il est donc nécessaire de se pencher sur la signification et les mécanismes de l'instanciation de composantes réutilisables.

4.2.3.1 Les mécanismes de l'instanciation

Pour se concentrer d'abord sur l'instanciation de composantes réutilisables, l'exemple de la grue, donné à la figure 4.3, se révèle utile car il ne contient pas d'élément de mémoire. La figure 4.11 donne un aperçu du mécanisme d'assemblage du sous-système de cette même grue (appelé ici `LRCrane`) à partir de l'instanciation de deux composantes élémentaires dont on retrouve la spécification en annexe D. La flèche de la grue (appelée `Boom`) utilise une instance de la composante élémentaire `Jack211b` et la pince de la grue (appelée `Hook`) utilise une instance de la composante élémentaire `SuctionCup`. Dans la figure 4.11 l'instanciation est implicite dans un énoncé de déclaration de variable, par exemple, `Boom : Jack211b`, définit la flèche de la grue.

D'après le modèle d'instanciation suggéré en section 3.1.2.1, une instance d'une composante réutilisable résulte d'une déclaration de variable dans un sous-système, par exemple `Boom : Jack211b`. Dans ce scénario le nom `Boom` définit alors un espace symbolique au sein duquel sont définis les éléments $\mathbf{G}$ et $\overline{\mathbf{G}}$ de la spécification de `Jack211b`. Par suite, pour référencer un de ces éléments, on doit faire usage de *noms qualifiés*. Par exemple, on peut présumer hypothétiquement que la déclaration précédente produise le holon

$$\mathcal{H}_{Boom} = (Q_{Boom}, \Sigma_{Boom}, \delta_{Boom}, \{q_{Boom,0}\}, Q_{Boom,m})$$

dans l'environnement du sous-système courant à partir de la définition de $\overline{\mathbf{G}}$ de la composante `Jack211b`.

On peut noter que le modèle d'instanciation de la section 3.1.2.1 est juste et produit le résultat attendu ; c'est-à-dire qu'il garantit que toutes les instances de composantes réutilisables d'un sous-système sont disjointes deux à deux. Bien sûr on présume que tous les

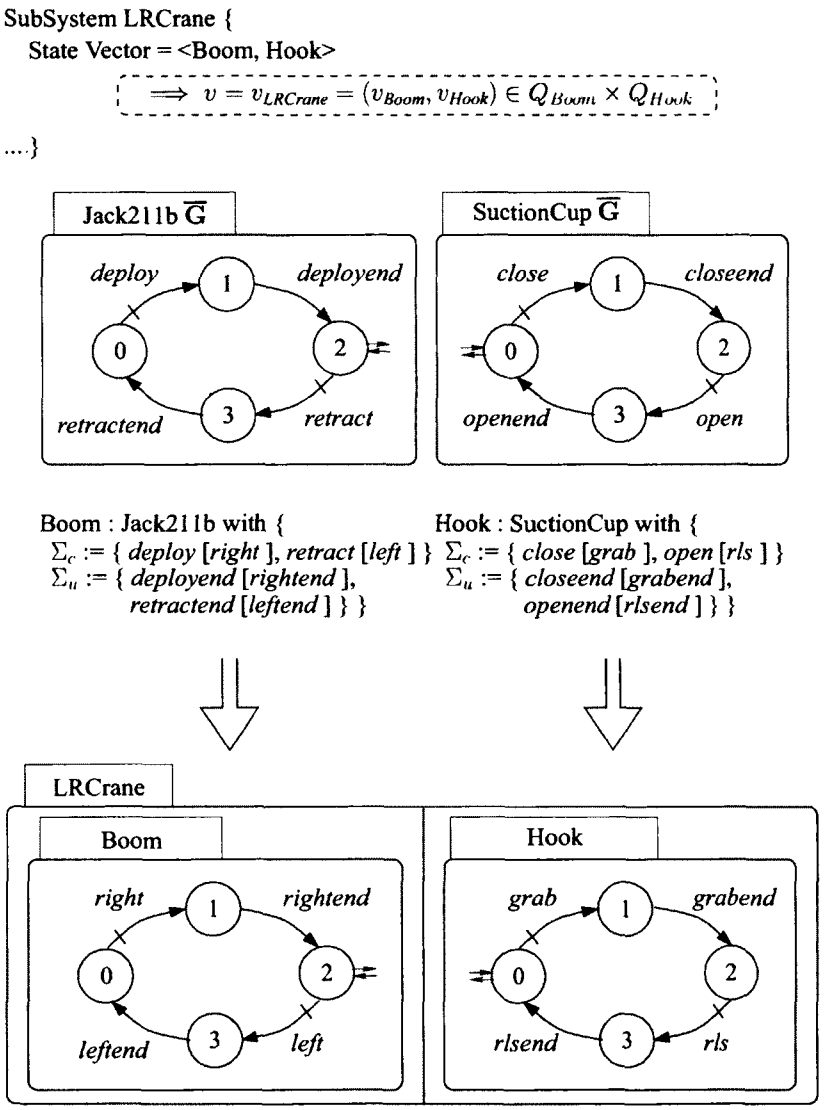


FIGURE 4.11 – Composition d'un sous-système (instanciation de composantes)

identificateurs de variables dans l'environnement d'un sous-système sont distincts. En particulier on obtient que l'événement $deploy \in T$ de la composante `Jack211b` est redéfini par $deploy_{Boom} \in Q_{Boom}$. Par conséquent la fonction $func_{deploy}()$, qui n'est pas définie dans la spécification de `Jack211b` (bien qu'elle y soit déclarée), se trouve automatiquement définie par association à la fonction $func_{deploy_{Boom}}()$ qui, elle, doit être générée automatiquement pour la dynamique abstraite du sous-système. Mais bien sûr la *qualification des noms*, bien qu'elle soit conceptuellement correcte, est beaucoup trop lourde pour qu'on l'utilise telle quelle dans la spécification. Aussi doit-on faire usage de quelques abus de notation ainsi que d'une convention de *changement de dénomination* (de l'anglais *aliasing*).

Tout d'abord à la figure 4.11 on retrouve un énoncé définissant la structure du vecteur d'état `State Vector = <Boom, Hook>`. Ainsi, dans l'environnement du sous-système `LRCrane`, il existe une variable $v_{LRCrane}$ dont la structure est un couple ordonné de la forme (v_{Boom}, v_{Hook}). On admet par convention que la définition de la structure du vecteur d'état donne aussi l'ordre de synthèse. Le premier abus de notation consiste à utiliser (lorsque le contexte est clair) le nom de la variable d'instance elle-même (par exemple `Boom`) en lieu et place de sa variable d'état (par exemple v_{Boom}) ; d'où la déclaration `State Vector = <Boom, Hook>`.

En deuxième lieu on peut noter qu'il est inutile de qualifier les étiquettes des états (qu'elles soient symboliques ou données par de simples index). La définition du *vecteur d'état* du sous-système permet toujours d'identifier le domaine des étiquettes d'état. Lorsque le sous-système de la figure 4.11 se trouve à l'état $(1, 2)$, alors le prédicat $v_{Boom} = 1 \wedge v_{Hook} = 2$ est réalisé. Ce prédicat s'énonce plus clairement par `Boom` = 1 $\wedge$ `Hook` = 2 en utilisant l'abus de notation signalé précédemment.

Finalement pour permettre d'éviter la qualification des étiquettes des événements, on suggère le mécanisme de *changement de dénomination* illustré en figure 4.11. Dans cette figure la déclaration complète pour la variable `Boom` est la suivante.

```
Boom : Jack211b with {
  Σc := { deploy[right], retract[left] }
  Σu := { deployend[rightend], retractend[leftend] }
}
```

Le concepteur doit dresser une bijection $h_{Boom} : T_{Jack211b} \rightarrow \Sigma_{Boom}$, où $T_{Jack211b}$ est

l'alphabet de projection de la composante `Jack211b` et Σ_{Boom} l'alphabet d'un *holon*,

$$\mathcal{H}_{Boom} = (Q_{Boom}, \Sigma_{Boom}, \delta_{Boom}, \{q_{Boom,0}\}, Q_{Boom,m})$$

construit à partir de l'*interface* $\overline{\mathrm{G}}$ de `Jack211b` et destiné à représenter une instance de cette composante dans l'environnement du sous-système. Le concepteur doit veiller à ce que les alphabets d'*alias* de toutes les instances au sein d'un même sous-système soient disjoints entre eux. Dans la figure 4.11 ceci veut dire que $\Sigma_{Boom} \cap \Sigma_{Hook} = \emptyset$. Par exemple le nom d'événement *right* utilisé par l'instance de composante `Boom` ne peut absolument pas être réutilisé par une autre instance dans ce sous-système ; il y est *unique* et désigne l'événement $deploy_{Boom}$ de l'interface d'une instance de la composante `Jack211b`. Par la même occasion la fonction $func_{deploy}()$, que l'on retrouve déclarée et utilisée dans la dynamique abstraite de la composante `Jack211b` est automatiquement associée à la fonction $func_{right}()$ générée pour le sous-système (une liaison statique). Ainsi, pour l'instance nommée `Boom`, tout appel à $func_{deploy}()$ dans la dynamique abstraite de `Jack211b` correspond à un appel de $func_{right}()$ liant ainsi la dynamique abstraite de la composante à celle du sous-système.

Avec cet ensemble de conventions en place on peut obtenir une spécification compacte de la structure du sous-système `LRCrane` correspondant à la figure 4.13. Combiné à la description des modèles de la projection G et de l'interface $\overline{\mathrm{G}}$, que l'on trouve à la figure 4.12, on obtient une spécification complète de la composante réutilisable `LRCrane`.

Dans la figure 4.12 on retrouve (au bas) un diagramme de l'assemblage du système. Ce diagramme n'est pas essentiel mais il constitue une documentation utile. D'ailleurs le diagramme de la figure 4.3 constitue une description détaillée des requis de contrôle de cette composante et le diagramme de la figure 4.4, une autre forme pour la spécification de la loi de contrôle obtenue par synthèse. Par opposition à la figure 4.13 ce genre de diagrammes peut présenter des avantages pour la clarté.

Concernant la dynamique abstraite de la composante `LRCrane`, il est clair que les fonctions $func_{\sigma}()$, $Project_{\sigma}()$ et $P_{\sigma}()$ peuvent être générées automatiquement, comme dans le cas des composantes élémentaires. Cependant, contrairement au cas des composantes élémentaires, les fonctions $lal_{\sigma}()$ ne peuvent pas être vides. Chaque instance de composante

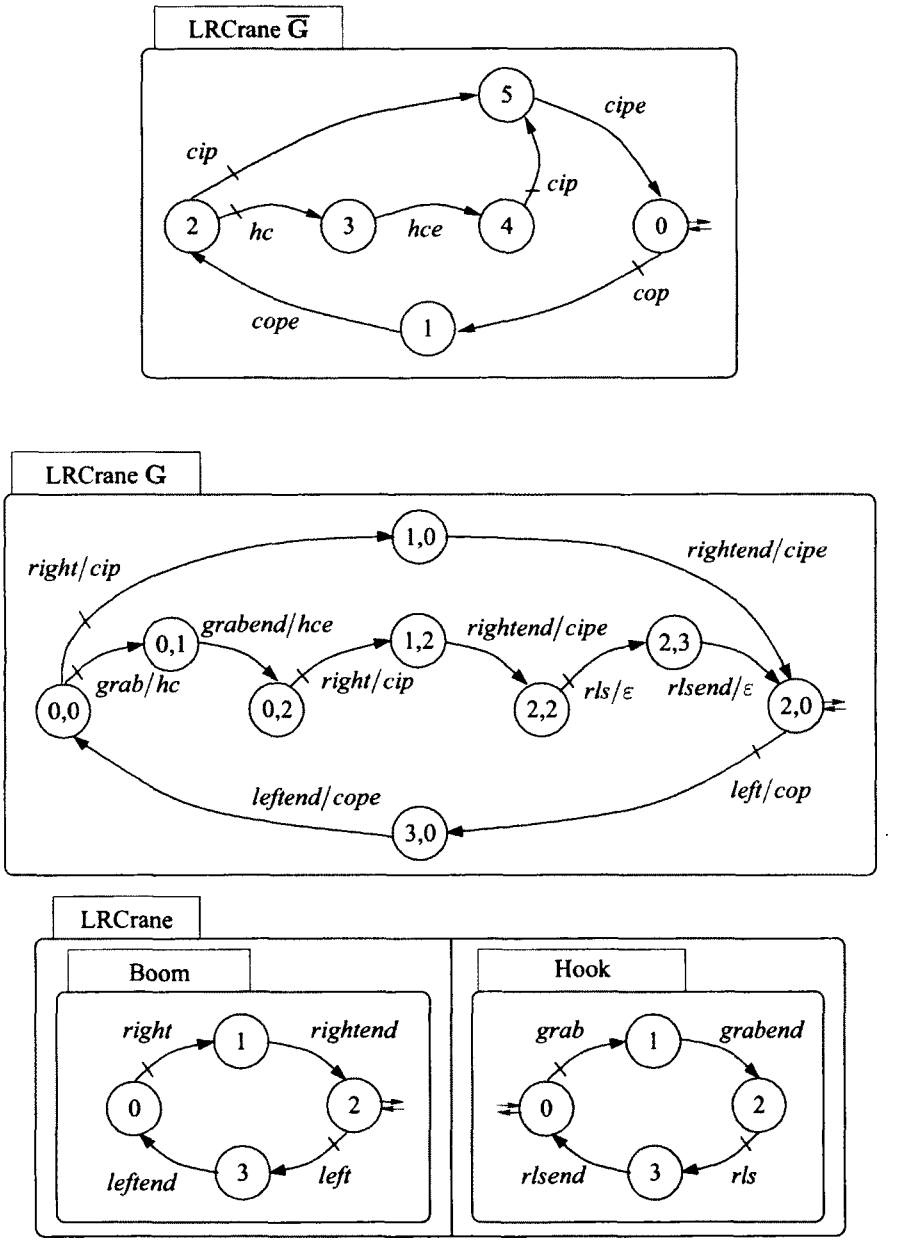


FIGURE 4.12 – Composante mixte *LRCrane* (sous-système et projection)

```
Component LRCrane {

  SubSystem LRCrane {
    State Vector = <Boom, Hook>

    Boom : Jack211b with {
      Σc := { deploy[right], retract[left] }
      Σu := { deployend[rightend], retractend[leftend] }
    }
    Hook : SuctionCup with {
      Σc := { close[grab], open[rls] }
      Σu := { closeend[grabend], openend[rlsend] }
    }

    CtrlSpec {
      Boom ∈ {1,3} ∧ Hook ∈ {1,3},
      Boom = 0 ∧ Hook = 3,
      Boom = 2 ∧ Hook = 1,
      Boom = 3 ∧ Hook = 2
    }

    f_right := Boom = 0 ∧ Hook ∈ {0,2}
    f_left  := Boom = 2 ∧ Hook = 0
    f_grab  := Boom = 0 ∧ Hook = 0
    f_rls   := Boom = 2 ∧ Hook = 2

  }
}
```

FIGURE 4.13 – Structure du sous-système *LRCrane*

réutilisable est associée à une variable d'état. Cette variable fait partie de l'environnement local du sous-système et doit être mise à jour pour refléter l'état courant de l'instance et, indirectement, l'état courant du sous-système lui-même. C'est l'une des tâches des fonctions $lal_\sigma()$.

Dans notre exemple de la figure 4.11, l'occurrence de l'événement *right* dans l'instance `Boom` doit provoquer le changement de la variable v_{Boom} de $v_{Boom} = 0$ à $v_{Boom} = 1$ pour maintenir l'intégrité de la variable d'état du sous-système $v_{LRCrane}$. Or l'information nécessaire pour générer ce code est déjà contenue dans la définition de la structure de transition de l'interface de la composante `Jack211b` ($\overline{G}_{Jack211b}$). Il s'agit de la fonction de transition $\eta_{Jack211b}$. La fonction $lal_{right}()$ est donc générée automatiquement et correspond à la définition suivante.

```
lal_right() := { Boom := 1; }
```

L'ensemble des fonctions $lal_\sigma()$ de notre exemple peuvent donc être générées automatiquement.

On peut présumer qu'en général, pour les composantes mixtes, les fonctions $lal_\sigma()$ sont générées automatiquement en conformité avec l'ensemble des définitions de la spécification et que ces fonctions ne sont pas vides. Le concepteur a tout de même la possibilité de redéfinir ce code ou d'insérer des fragments de code supplémentaires soit au début soit à la fin d'une fonction $lal_\sigma()$ en utilisant les notations mentionnées à la fin de la section 4.2.2. Naturellement, comme pour les composantes élémentaires, les symboles pour l'interface des composantes mixtes (les fonctions $func_\tau()$) sont toujours des symboles *abstraits* (*déclarés sans définition* dans le sous-système de la composante). Aussi l'heuristique de génération de code *cycle*() demeure disponible mais elle n'a pas de contenu *a priori*. Dans la composante `LRCrane` par exemple, cette heuristique est vide.

Finalement on peut constater que la composante réutilisable `LRCrane`, en figure 4.13, n'a pas de signature d'instanciation. Elle utilise cependant deux composantes réutilisables qui requièrent la spécification éventuelle de paramètres ; `Jack211b` et `SuctionCup`. Ceci ne constitue pas une contradiction.

Comme mentionné en section 4.2.1.2 les signatures d'instanciation ne servent qu'à spécifier l'*assignation* de ressources réelles à des symboles génériques utilisés dans la spécification de composantes. Par exemple un capteur peut être représenté par le symbole X0 pour

la spécification sans qu'il soit nécessaire de déterminer immédiatement à quelle ressource physique ce capteur correspond. Ces assignations ne sont pas requises pour la conception et la synthèse. Elles sont requises *a posteriori* pour la génération de code. Il est alors possible de les spécifier au générateur de code dans une spécification indépendante en utilisant des noms qualifiés complets. Par exemple à supposer que le sous-système `LRCrane` de la figure 4.11 soit considéré comme un *système final* et non pas comme le sous-système associé à une composante réutilisable. Alors la spécification de ses assignations peut prendre la forme suivante.

```
LRCrane.Boom<Y0 := Q0.0, Y1 := Q0.1,
             X0 := I0.0, X1 := I0.1>;
LRCrane.Hook<Y0 := Q1.0, Y1 := Q1.1, X0 := I1.1>;
```

On évite ainsi d'avoir à paramétrer le système au fur et à mesure de la conception. D'autres mécanismes peuvent éventuellement être mis en place.

4.2.3.2 Composante mixte avec mémoire incorporée

La composante réutilisable `LRCrane` n'utilise aucun élément de mémoire dans son sous-système. La structure de sa spécification est par conséquent très simple. Il est cependant souvent très utile sinon nécessaire de faire usage d'éléments de mémoire pour décrire les requis de contrôle d'un sous-système, entre autres lorsque certains aspects du contrôle dépendent de l'*historique* de fonctionnement du sous-système.

La figure 4.14 donne le schéma d'assemblage d'une perceuse. Le sous-système (appelé ici *Drill*) comporte plusieurs éléments dont on doit coordonner l'activité.

1. Un étau (*Clamp*) assure, par serrage, la stabilité de la *pièce* pendant l'opération de perçage.
2. Un moteur électrique (*Mtr*) contrôlé par un relais actionne la mèche de la perceuse.
3. Le moteur est fixé sur un patin (*Drv*) qui peut descendre et monter pour effectuer le perçage.

La machine a deux états principaux (éléments de mémoire SW) elle est soit en marche, soit à l'arrêt. L'opération de perçage (une fois la machine en marche) se déroule en deux phases (élément de mémoire PHS).

1. La pièce est d'abord stabilisée, le moteur démarré et le patin descendu puis désengagé (SW = 1 et PHS = 0).
2. Ensuite le moteur est arrêté et l'étau ouvert (SW = 1 et PHS = 1).

La machine doit alors être arrêtée. L'ensemble doit donner un comportement global de type *démarrage-arrêt* (élément de mémoire SW). Le tout est encapsulé en une composante réutilisable (*Drill*) dont l'abstraction correspond à la figure 4.15, la spécification de structure à la figure 4.16 et la spécification de contrôle à la figure 4.17.

L'ensemble des instances du sous-système correspond à des composantes élémentaires de l'inventaire en annexe D. La composante `Jack211aSS` (pour le patin) n'y figure pas, mais il est aisé de la produire à partir de la composante `Jack211a`. Il suffit d'utiliser le même schéma que celui suivi pour élaborer la composante `Jack111SS` à partir de la composante `Jack111`.

Par rapport à la spécification de la composante `LRCrane`, discutée précédemment, on peut noter deux additions dans la spécification de la composante `Drill`. Tout d'abord on y fait usage d'éléments de mémoire (SW et PHS dans figure 4.14). Aussi on détecte la présence d'une nouvelle classe d'événements, les événements dits *synthétiques*, au bas de la figure 4.16. Ces additions requièrent quelques explications.

L'ajout d'éléments de mémoire ne présente pas de problème spécifique. Bien sûr chaque élément de mémoire doit être représenté dans l'environnement du sous-système par une variable d'état, par exemple v_{PHS} pour l'élément de mémoire dénotant les phases de l'opération de perçage. Cette variable est représentée par une déclaration dans la définition de la

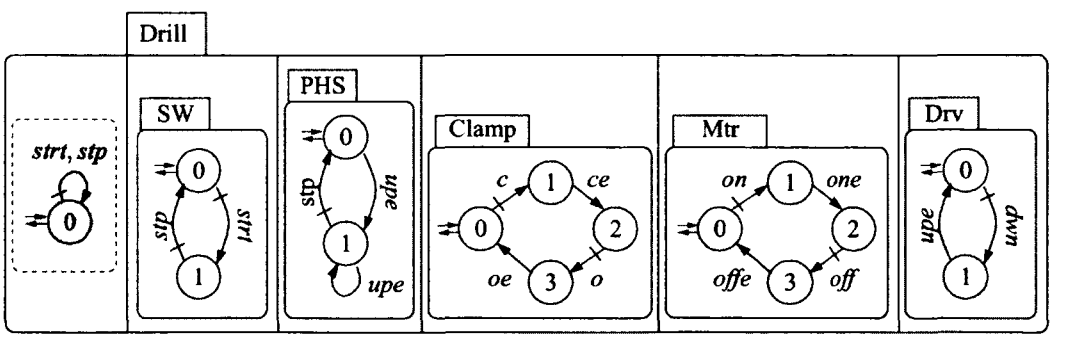


FIGURE 4.14 – Composante mixte *Drill* (assemblage du sous-système)

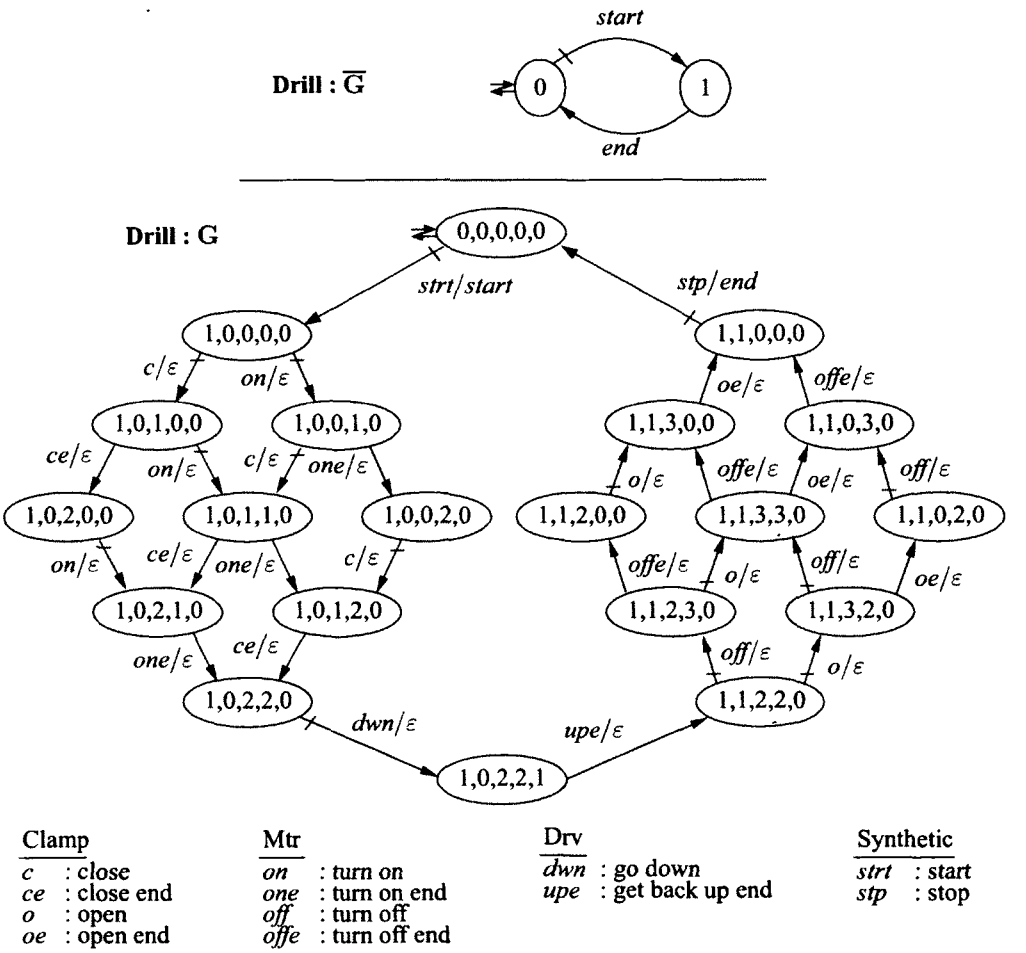


FIGURE 4.15 – Composante mixte *Drill* (sous-système et projection)

```
Component Drill {

  SubSystem Drill {
    State Vector = <SW, PHS, Clamp, Mtr, Drv>

    SW   : MW; /* Memory word : Drilling SWitch */
    PHS  : MW; /* Memory word : Drilling PHaseS */

    Clamp : Jack111 with {
      Σc := { deploy[c], retract[o] }
      Σu := { deployend[ce], retractend[oe] }
    }
    Mtr : SDRelay with {
      Σc := { close[on], open[off] }
      Σu := { closeend[one], openend[offe] }
    }
    Drv : Jack211aSS with {
      Σc := { start[dwn] }
      Σu := { stop[ue] }
    }
    Synthetic Events := {
      Σc := { strt, stp }
    }
  ...
```

FIGURE 4.16 – Sous-système *Drill* (spécification de structure)

```
Component Drill {
  SubSystem Drill {
    State Vector = <SW, PHS, Clamp, Mtr, Drv>
    ...
    CtrlSpec {
    /*(1)*/ SW = 0 ∧
            (Clamp ∈ {1,2,3} ∨ Mtr ∈ {1,2,3} ∨ Drv = 1),
    /*(2)*/ PHS = 0 ∧ (Clamp = 3 ∨ Mtr = 3),
    /*(3)*/ PHS = 1 ∧ (Clamp = 1 ∨ Mtr = 1),
    /*(4)*/ Drv = 1 ∧ PHS = 1,
    /*(5)*/ Drv = 1 ∧ (Clamp ∈ {0,1,3} ∨ Mtr ∈ {0,1,3})
    }

    f_strt := SW = 0
    f_stp := PHS = 1 ∧ Clamp = 0 ∧ Mtr = 0
    f_c := SW = 1 ∧ PHS = 0
    f_o := SW = 1 ∧ PHS = 1
    f_on := SW = 1 ∧ PHS = 0
    f_off := SW = 1 ∧ PHS = 1
    f_dwn := SW = 1 ∧ PHS = 0 ∧ Clamp = 2 ∧ Mtr = 2

    cycle() := cycle() + {
      func_strt();
      func_stp();
    }
  } }
```

FIGURE 4.17 – Sous-système *Drill* (spécification du contrôle)

structure du sous-système (`PHS : MW` en figure 4.16). Cette variable peut donc être utilisée de la même façon que la variable d'état d'une instance de composante réutilisable et faire l'objet d'affectation (`PHS := 1`) ou dénoter un terme dans un prédicat (voir la figure 4.17).

Au sein d'un sous-système, les éléments de mémoire sont représentés par des AFD dont tous les événements sont *partagés* avec les autres éléments du sous-système. Leur dynamique (la fonction de transition de leur AFD) doit être explicite pour permettre la synthèse. À cet effet on adopte les conventions utilisées dans [39]. L'AFD d'un modèle de mémoire doit être *fermé* sur l'ensemble des événements qu'il utilise dans sa dynamique.

Pour les événements contrôlables, toutes les transitions doivent être explicites. L'absence d'une transition à partir d'un état constitue une contrainte de contrôle explicite. Par exemple, pour l'élément PHS en figure 4.14, l'absence d'une transition sur l'événement *stp* à partir de l'état 0 constitue une contrainte de contrôle. Aussi chaque état de l'AFD doit comporter une transition pour chaque événement incontrôlable partagé par cet élément de mémoire (la *fermeture* de l'AFD). Lorsqu'une transition incontrôlable ne doit pas se produire dans le système contrôlé, cette transition (sur événement incontrôlable) doit faire l'objet d'une *contrainte* dans la *spécification de contrôle*. Par exemple, pour l'élément PHS en figure 4.14, la transition en boucle sur l'événement *upe* à l'état 1 est indésirable puisque le patin (*Drv*) de la perceuse ne doit pas être actif (`Drv = 1`) une fois le perçage terminé (`PHS = 1`). Ainsi il faut introduire la contrainte dénotée (4) dans la spécification de contrôle de la figure 4.17.

Il est intéressant de noter, par exemple, qu'en figure 4.14 l'AFD PHS ne partage pas l'événement *ce* généré par *Clamp*. Or pourtant la spécification de contrôle stipule bien une contrainte d'exclusion entre les états `PHS = 1` et `Clamp = 1` (contrainte (3) en figure 4.17). Cette situation est fréquente. Il faut noter que, puisque PHS ne partage pas l'événement *ce* généré par *Clamp*, il y a une transition en boucle *implicite* sur l'événement *ce* dans l'AFD PHS à partir chacun de ses états. La contrainte (3) vise ainsi à exclure du comportement contrôlé la transition en boucle *implicite* sur l'événement *ce* se produisant à l'état 1 de l'AFD PHS. Autrement dit le *support* des contraintes de contrôle se situe souvent dans le comportement *implicite* au modèle du sous-système. Cette situation est fréquente comme en font foi les spécifications de contrôle des composantes *Drill* (en figure 4.17)

et *LRCrane* (en figure 4.13). Heureusement cette forme de spécification par exclusion de familles d'états du comportement contrôlé est habituellement assez *intuitive*. Pour s'en convaincre on peut consulter l'explication des contraintes pour la conception d'une grue (à la section 3.2.2) qui sont essentiellement les mêmes que celles utilisées pour la spécification de la composante `LRCrane`.

Il faut maintenant clarifier la notion d'*événements synthétiques*. Il s'agit d'une technique introduite par Leduc dans [34] pour permettre la modélisation en HISC de certaines situations où il est nécessaire d'ajouter des événements au système. Ces événements n'ont généralement pas de base factuelle et ne sont pas générés par des dispositifs physiques. On peut décrire les événements *synthétiques* comme des événements générés par le sous-système lui-même dans le but d'en modifier l'état de la mémoire. On peut parler d'événements liés à des mouvements d'éléments de mémoire. L'utilisation d'éléments de mémoire est donc implicite à l'usage de cette technique.

Dans le modèle théorique on adjoint au sous-système un AFD comportant *un et un seul* état avec une transition en boucle sur cet état pour chacun des événements de son alphabet. À la figure 4.14, cet AFD est représenté par une boîte grisée à gauche de la figure. Puisque cet AFD a toujours la même forme et qu'il n'ajoute aucune information d'état au modèle, il n'est normalement pas représenté. Il suffit de donner une liste des événements synthétiques dans la spécification de structure comme en figure 4.16. Cet AFD représente cependant un *dispositif* au même titre que les instances de composantes réutilisables. Il ne s'agit donc pas d'un AFD d'élément de mémoire. Ainsi cet AFD a un *alphabet disjoint* de tous les autres *dispositifs*. Il ajoute donc au sous-système un certain nombre d'événements (habituellement des événements contrôlables). L'occurrence de n'importe lequel de ces événements modifie nécessairement l'état d'*au moins un* des éléments de mémoire interne à un sous-système.

Avec ce modèle théorique les événements synthétiques entrent *naturellement* dans la modélisation, la spécification de contrôle et la synthèse de sous-systèmes au même titre que n'importe quel autre dispositif. Par exemple, pour le sous-système *Drill* illustré à la figure 4.14, il y a deux événements synthétiques, *strt* et *stp*. Ces deux événements sont partagés par l'AFD SW, et *stp* est aussi partagé par l'AFD PHS, exprimant ainsi des *contraintes de contrôle explicites* puisque ces événements sont contrôlables. Aucune des instances de composantes réutilisables du sous-système ne partage ces événements (en conformité avec

le modèle théorique). Il y a donc des transitions en boucle implicites à tous les états des modèles d'AFD pour *Clamp*, *Mtr* et *Drv* et ce, pour chacun des événement *strt* et *stp*. Aussi retrouve-t-on, entre autres, la contrainte (1) en figure 4.17 pour interdire l'occurrence de l'événement *strt* (c'est-à-dire le démarrage du sous-système) à moins que tous les *dispositifs* (*Clamp*, *Mtr* et *Drv*) ne soient à l'état initial.

Concernant la génération de code, la synthèse procure les prédicats de contrôle f_{strt} et f_{stp} et donc la définition de $P_{strt}()$ et $P_{stp}()$. Les fonctions $func_{strt}()$ et $func_{stp}()$ peuvent être générées en conformité avec l'algorithme 4.1 (événements contrôlables) et les fonctions $Project_{strt}()$ et $Project_{stp}()$ en conformité avec l'algorithme 4.2. Aussi, puisque la fonction de transition de tous les AFD est donnée explicitement, la mise à jour des variables d'état pour les instances de composantes réutilisables ainsi que pour les éléments de mémoire peut être générée automatiquement dans les fonctions $lal_{strt}()$ et $lal_{stp}()$.

```
lal_strt() := { SW := 1; }
lal_stp() := { SW := 0; PHS := 0; }
```

Comme pour les composantes élémentaires, il est nécessaire de lier la dynamique abstraite du sous-système à l'occurrence des événements *strt* et *stp* puisque ces derniers sont générés par le sous-système lui-même. À cette fin, comme illustré en figure 4.17, on introduit les appels correspondants dans l'heuristique *cycle*().

On peut remarquer qu'à la figure 4.15 l'événement *stp* (qui est contrôlable dans le sous-système) est projeté sur l'événement d'interface *end* qui est incontrôlable. La règle de l'équation C.5 pour la conception d'interface (calcul d'un *observateur*) permet en effet cette manoeuvre (mais pas l'inverse bien sûr).

D'ailleurs les mécanismes d'instanciation présentés dans cette section permettent aussi de changer, à l'instanciation, le statut d'un événement contrôlable de l'interface pour le considérer incontrôlable. La déclaration suivante

```
Boom : Jack111 with {
  Σc := { deploy[right] }
  Σu := { retract[left],
          deployend[rightend],
          retractend[leftend] }
}
```

a précisément cet effet. L'événement d'interface *retract* de la composante `Jack111`, qui est originalement contrôlable, est déclaré comme l'événement incontrôlable *left* dans le sous-système qui instancie la composante. Cette déclaration a pour effet de forcer le prédicat de contrôle de l'événement *left*, f_{left}, en tautologie.

Finalement, il est parfois plus clair d'exprimer explicitement certaines contraintes de contrôle par partage d'événements contrôlables entre les modèles. La figure 4.18 offre une version alternative du diagramme d'assemblage de la composante *Drill* où certaines contraintes sont exprimées directement sur les AFD des instances du sous-système. Dans ce diagramme on voit que l'élément de mémoire SW a été supprimé et que l'ensemble de ses événements (tous contrôlables) sont maintenant partagés par un ou plusieurs des autres éléments (AFD) du modèle. Ceci est le seul cas où des événements synthétiques peuvent être privés du support d'un élément de mémoire. Avec ce nouveau diagramme d'assemblage, la contrainte de contrôle dénotée (1) dans la figure 4.17 devient inutile puisqu'elle fait maintenant partie intégrante de la structure de transition du sous-système.

Dans un modèle de sous-système, rien n'empêche de *partager directement les événements contrôlables* entre les AFD d'instances de composantes pour exprimer des contraintes de contrôle. Il suffit de disposer d'une notation appropriée pour l'exprimer (ici le diagramme d'assemblage). Cette façon de procéder peut être plus compacte lorsque les contraintes exprimées sont claires et *n'impliquent que des transitions sur événements contrôlables*. Il faut noter que cette façon d'exprimer des contraintes *modifie explicitement la structure de transition* des composantes impliquées, dans ce cas le diagramme de structure

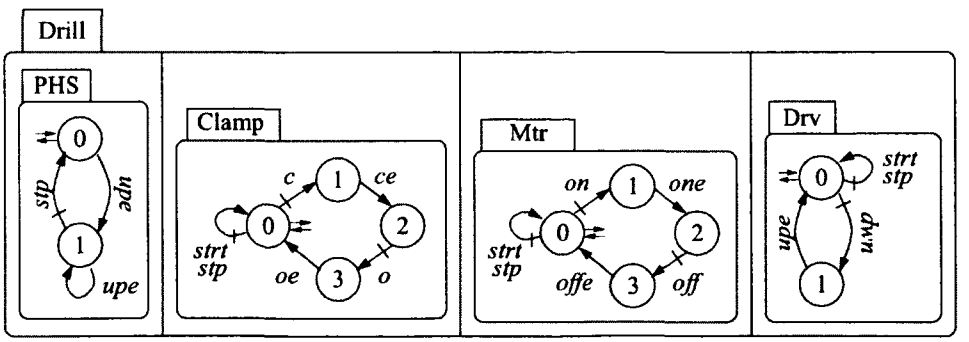


FIGURE 4.18 – Composante *Drill* (diagramme d'assemblage alternatif)

devient un élément essentiel de la spécification de contrôle. Une alternative consiste à donner les transitions à ajouter explicitement dans la spécification de contrôle ou dans l'énoncé d'instanciation des composantes réutilisables. La technique peut entraîner une tendance à la *surspécification* mais elle est trop commode pour s'en passer.

4.3 Processus global de conception

Le processus de conception, tel que préconisé par le genre de composantes réutilisables considérées ici, consiste en un processus de raffinements successifs :

1. concevoir un sous-système, ce qui veut dire formuler et résoudre un problème de contrôle ;
2. abstraire le résultat de la conception d'un sous-système (la solution à un problème de contrôle) en lui adjoignant une interface appropriée à l'aide de l'algorithme de Wong ;

de bas en haut, jusqu'à l'obtention d'un système global. Bien sûr il y *une* exception : le système global lui-même. Mais même le système global peut être considéré comme une composante dont l'interface efface tous les événements.

Le processus de conception d'un sous-système prend deux formes différentes selon qu'il s'agit du sous-système d'une composante de base (où le processus de conception est réalisé de façon *ad hoc*) ou de celui d'une composante mixte (conception SCT à partir d'instances de composantes réutilisables). Mais ces deux types de processus se déroulent selon un schéma commun et doivent produire des spécifications similaires.

1. On doit obtenir un modèle du comportement libre du sous-système qui sert de base à l'élaboration de la spécification de contrôle.
2. Ensuite on procède à l'élaboration de la spécification de contrôle.
3. Finalement on procède à l'extraction de la loi de contrôle correspondant à la spécification de contrôle et à l'élaboration d'un modèle du système sous contrôle.

Pour le sous-système d'une composante mixte, le processus est conforme à SCT. Il faut établir les éléments suivants :

1. le diagramme d'assemblage ;
2. la spécification de structure comprenant : l'inventaire des ressources du sous-système, la spécification de contrôle et la loi de contrôle ;

(voir en section D.2). La loi de contrôle est naturellement obtenue par application des procédures de synthèse de SCT. L'expression du système sous contrôle local ($\mathbf{G_{ts}}$) n'est nécessaire que si le sous-système doit être doté d'une interface (c'est-à-dire abstrait).

Pour le sous-système d'une composante de base, puisque le processus est *ad hoc*, tous les éléments sont spécifiés *à priori* et documenté selon la méthode choisie. Le modèle du comportement libre prend en général la forme d'un modèle exhaustif du système (voir section 3.1.1.2). La loi de contrôle est spécifiée *a priori* par insertion de code d'intégration donné dans la spécification de structure (voir section D.1). Finalement un modèle exhaustif est dressé *à priori*. Certaines lignes directrices sont utiles comme : dresser des AFD comme modèles, associer chaque état de ces AFD à une configuration détectable du matériel et enfin de baser l'alphabet d'événements sur des changements détectables de la configuration du matériel. Le modèle exhaustif doit nécessairement être un AFD puisqu'il sert de point d'entré à l'abstraction d'une interface (le modèle $\mathbf{G_{ts}}$). L'usager est responsable de vérifier formellement ou, au moins, de valider statistiquement le comportement sous contrôle présumé. Il en va de la qualité des modèles qui plus tard sont soumis à la synthèse.

Le processus d'abstraction, lui, est commun au deux type de composantes et se déroule toujours selon le même schéma. À partir d'un modèle du sous-système sous contrôle local $\mathbf{G_{ts}}$,

1. on établit une machine de Mealy pour la projection désirée (il s'agit d'hypothèses que l'on fait sur les transitions à projeter) ;
2. on soumet ces hypothèses de projection à l'algorithme de Wong qui calcule une bisimulation du sous-système et, le cas échéant, soumet des hypothèses de projection supplémentaires ;
3. si on est satisfait du résultat, on a terminé, sinon on doit reformuler des hypothèses et boucler à l'étape (2).

À la sortie de cette procédure heuristique, le résultat de l'algorithme de Wong est une définition d'interface adéquate (c'est-à-dire une projection dotée de la propriété d'observateur).

Chapitre 5

Étude de cas, le *MPS*

L'usine école *MPS* est une usine modulaire ***Festo***^{TM} dont on peut trouver la description détaillée dans l'ensemble des brochures techniques : [17, 18, 19, 20] et [21]. Cette usine modulaire est conçue spécifiquement pour la formation de techniciens ayant à programmer ce type de dispositifs en industrie. L'usine école *MPS* constitue donc un exemple représentatif de dispositif industriel dont la taille procure une étude de cas représentative du type de problèmes rencontrés. Dans son ensemble, d'une douzaine de *cellules de traitement*, l'usine école matérialise le processus d'assemblage de pistons pour des moteurs à combustion. Cependant, puisqu'il s'agit d'un outil pédagogique, les matériaux utilisés ne sont pas les matériaux réels, question de coûts, et les éléments *usinés* (des pièces rondes en plastique et en aluminium) peuvent être réutilisés.

L'usine école *MPS* comporte potentiellement une douzaine de cellules de traitement divisées en plusieurs stations. Le laboratoire des système réactifs n'en compte que quatre (les quatre premières) :

1. la station de livraison ;
2. la station de test ;
3. la station d'usinage ;
4. la station de manutention.

Dans le montage du laboratoire l'ensemble est contrôlé par un automate programmable

Siemens^{TM} modèle 315PN/DP [51] sur un réseau ***Profibus***^{TM}.

L'étude de cas proposée s'attache à formaliser et résoudre les problèmes de contrôle des stations de livraison, de test et d'usinage. Pour pouvoir modéliser un système complet, la grue de transfert à la sortie de la station d'usinage est incluse dans le sous-système modélisant le processus d'usinage. L'étude de cas modélise donc trois sous-systèmes :

1. le sous-système *DStn* pour la station de livraison ;
2. le sous-système *TStn* pour la station de test ;
3. le sous-système *PStn* pour la station d'usinage comprenant la grue de transfert.

Une spécification informelle des trois sous-systèmes, sous forme de *grafcets*, est donnée dans l'ensemble des brochures ***Festo***^{TM} citées précédemment. Cette description informelle n'est pas complète mais constitue une base suffisante pour élaborer les spécifications de l'étude de cas.

Pour fin d'illustration les sous-systèmes *DStn*, *TStn* et *PStn* sont d'abord modélisés comme des composantes *locales* du projet. Leurs problèmes de contrôle respectifs sont ainsi spécifiés et résolus indépendamment, puis leurs solutions respectives sont abstraites par projection comme dans le cas de composantes réutilisables. Un système global est ensuite assemblé à partir des interfaces de ces composantes locales, son problème de contrôle spécifié et, enfin, résolu par synthèse. Cette manoeuvre, bien qu'elle ne soit pas essentielle, permet de diviser le problème de contrôle global en sous-problèmes de contrôle d'envergure plus modeste, plus faciles à comprendre et à spécifier. La modélisation s'effectue à partir du catalogue des composantes réutilisables (élémentaires et mixtes) élaboré à l'annexe D.

5.1 Sous-système DStn, la station de livraison

La station de livraison du *MPS* a comme fonction d'alimenter le processus d'usinage en présentant des *pièces* à usiner (des rondelles de plastique et d'aluminium) une à une à la station de test située en aval. Elle est composée de deux éléments :

1. un distributeur de pièces (l'*injecteur*) modélisé à partir de la composante élémentaire `Injector` ;
2. une *grue de transbordement* modélisée par la composante mixte `BLRCrane`.

On trouve en figure 5.1 un diagramme d'assemblage des éléments du problème de contrôle de la station de livraison avec le détail de son séquenceur (un élément de mémoire) en figure 5.2. La figure 5.3 donne le modèle d'interface $(G,\overline{G})$ de la solution (la composante locale *DStn*).

La version abstraite du sous-système sous contrôle, illustrée en figure 5.3, est obtenue avec la définition de la fonction de projection donnée dans le tableau 5.1. La structure de transition de la solution est un peu trop volumineuse pour l'illustrer par une figure (34 états et 47 transitions) aussi préfère-t-on ici un tableau. Dans le tableau, seules les transitions pertinentes pour la projection sont données. Il est présumé que la projection efface toutes les autres transitions. Le modèle de la projection $\overline{G}$ de la figure 5.3 est un *observateur* obtenu par application de l'algorithme de Wong et Wonham [60] en faisant certaines hypothèses sur la structure de transition de la solution du problème de contrôle.

Informellement, l'injecteur distribue les pièces à usiner sur un dock de chargement situé à gauche de la grue. Une fois au dock, une pièce doit y être maintenue par l'injecteur jusqu'à ce que la grue de transbordement la saisisse. L'injecteur doit alors relâcher son emprise sur la pièce pour que la grue puisse la transborder jusqu'à la plateforme de la station de test située à droite de la grue. La position initiale de la grue (au repos, lorsque l'injecteur est vide) est à la plateforme de la station de test (lorsque cette dernière est aussi vide bien sûr). Il y a cependant une interaction nuisible entre la grue et la station de test. La grue ne peut fonctionner que si la station de test est au repos (plateforme de test vide, rien à tester). Réciproquement, la station de test ne peut fonctionner que si la grue est en position au dock de chargement.

La plateforme de la station de test est munie d'un capteur de présence. Cependant ce dernier ne peut pas être utilisé par la station de livraison puisqu'il n'en fait pas partie. Le *séquenceur* incorpore donc une mémoire pour se *rappeler* chaque fois qu'il a livré une pièce à la station de test. À chaque livraison d'une pièce la grue de la station de livraison est systématiquement rapatriée au dock de chargement. Elle n'est ensuite *libérée* qu'à travers un événement du sous-système ; l'événement synthétique contrôlable *rst*. Ainsi la station de livraison peut être contrôlée pour attendre que la station de test fasse son travail avant de livrer une autre pièce ou de regagner sa position initiale le cas échéant.

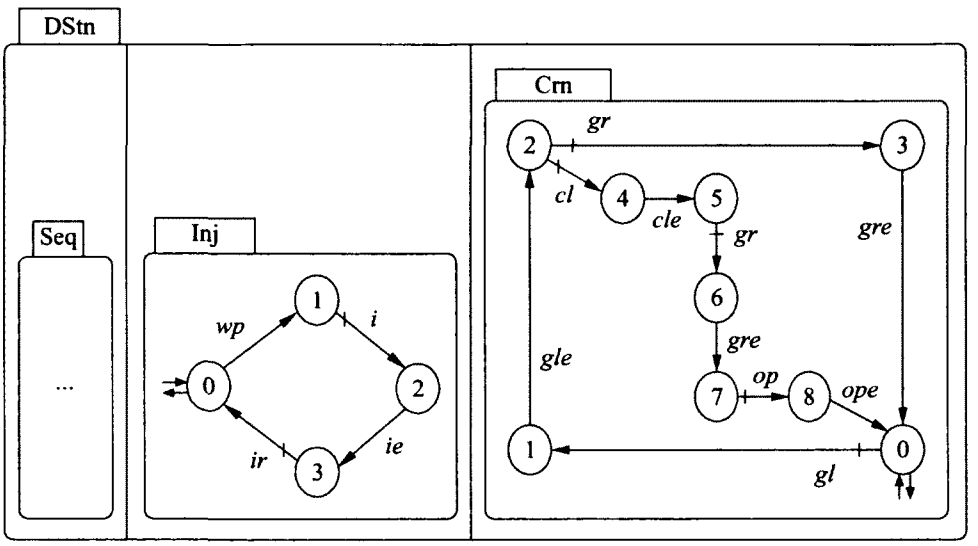


FIGURE 5.1 – Sous-système *DStn*, diagramme d'assemblage

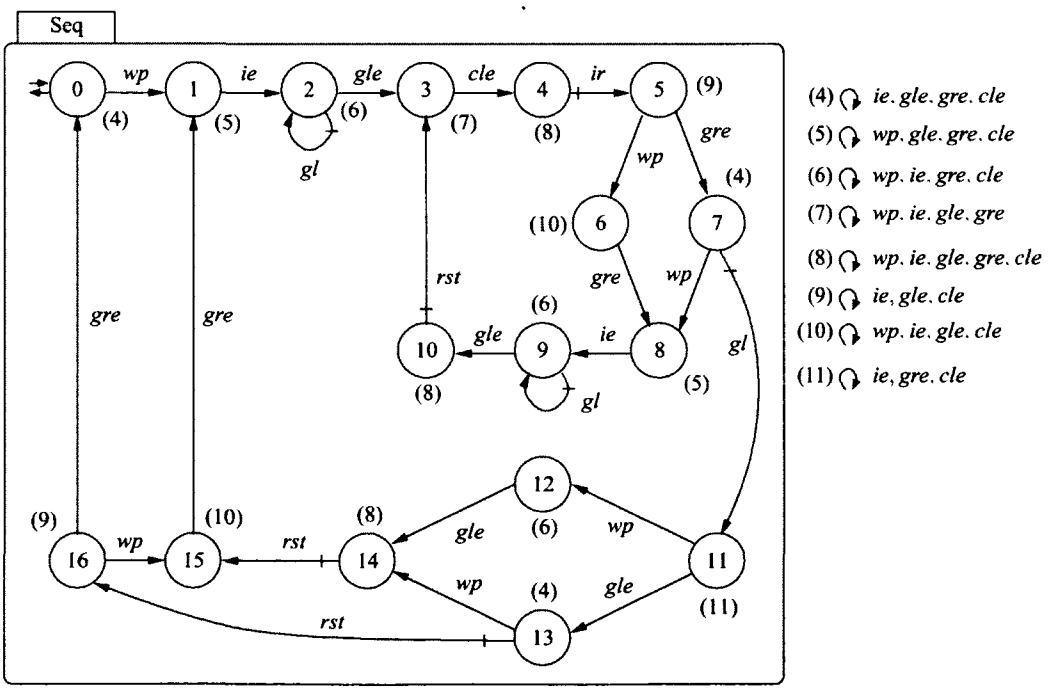
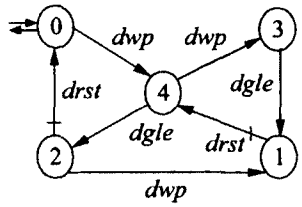


FIGURE 5.2 – Sous-système *DStn*, détails du séquenceur

DStn : $\overline{G}$

dwp : arrivée d'une pièce
dgle : arrivée au dock
drst : verrouillage au dock



DStn : G

Inj		Crn			
wp	: work piece arrival	*gl*	: go left	*cl*	: close
i	: inject	*gle*	: go left end	*cle*	: close end
ie	: inject end	*gr*	: go right	*op*	: open
ir	: injector rearm	*gre*	: go right end	*ope*	: open end

Voir le tableau 5.1 pour la définition de la projection.

FIGURE 5.3 – Sous-système *DStn*, et projection

TABLE 5.1 – Sous-système *DStn*, modèle de la projection **G**

$q \in Q_{DStn}$	$\sigma \in \Sigma_{DStn}$	$\lambda_{DStn}(q,\sigma)$	$q \in Q_{DStn}$	$\sigma \in \Sigma_{DStn}$	$\lambda_{DStn}(q,\sigma)$
0,0,0	*wp*	*dwp*	7,0,8	*wp*	*dwp*
11,0,1	*wp*	*dwp*	11,0,1	*gle*	*dgle*
13,0,2	*wp*	*dwp*	12,1,1	*gle*	*dgle*
16,0,2	*wp*	*dwp*	2,3,1	*gle*	ε
16,0,3	*wp*	*dwp*	9,3,1	*gle*	*dgle*
5,0,5	*wp*	*dwp*	10,3,2	*rst*	*drst*
5,0,6	*wp*	*dwp*	13,0,2	*rst*	*drst*
7,0,0	*wp*	*dwp*	14,1,2	*rst*	*drst*
7,0,7	*wp*	*dwp*			

Les détails structurels de la composante locale *DStn*, comprenant la spécification formelle de ses requis de contrôle ainsi que la formulation de la loi de contrôle obtenue par synthèse sur l'ensemble des contraintes, sont donnés dans la spécification de composante qui suit.

```
Component DStn {

  SubSystem DStn {
    State Vector = <Seq, Inj, Crn>

    Seq : MW; /* Sequencer memory 0..16 */
    Inj : Injector with {
      Σc := { inject[i], rearm[ir] }
      Σu := { injectend[ie], wpa[wp] }
    }
    Crn : BLRCrane with {
      Σc := { cbl[gl], cbr[gr], chc[cl], cho[op] }
      Σu := { cble[gle], cbre[gre], chce[cle], choe[ope] }
    }
    Synthetic := { Σc := { rst } }

    CtrlSpec {
    /* Implicit */
    /*(1) */ Inj = 2 ∧ Crn ∈ {1,2,3,4,5,6} (Safety)
    /*(2) */ Inj = 3 ∧ Crn ∈ {3,6} (Safety)
    /*(3) */ Inj ∈ {0,1,2} ∧ Crn = 4 (Safety + fct)
    /* Standard wrt Seq */
    /*(4) */ Seq ∈ {0,7,13} ∧ ( Inj = 2 ∨ Crn ∈ {1,3,4,6})
    /*(5) */ Seq ∈ {1,8} ∧ ( Inj = 0 ∨ Crn ∈ {1,3,4,6})
    /*(6) */ Seq ∈ {2,9,12}
              ∧ ( Inj ∈ {0,2} ∨ Crn ∈ {3,4,6})
    /*(7) */ Seq = 3 ∧ ( Inj ∈ {0,2} ∨ Crn ∈ {1,3,6})
    /*(8) */ Seq ∈ {4,10,14}
              ∧ ( Inj ∈ {0,2} ∨ Crn ∈ {1,3,4,6})
    /*(9) */ Seq ∈ {5,16} ∧ ( Inj = 2 ∨ Crn ∈ {1,4})
    /*(10)*/ Seq ∈ {6,15} ∧ ( Inj ∈ {0,2} ∨ Crn ∈ {1,4})
    /*(11)*/ Seq = 11 ∧ ( Inj = 2 ∨ Crn ∈ {3,4,6})
    /* Explicit wrt Seq */
```

```
    /*(12)   gl  <= Seq ∈ {2,7,9} */
    /*(13)   ir  <=  Seq = 4 */
    /*(14)   rst <= Seq ∈ {10,13,14} */
    }

    f_rst :=
      Seq ∈ {10,13,14}
    f_i :=
      Inj = 1 ∧ Seq ∈ {1,8}
    f_ir :=
      Inj = 3 ∧ Seq = 4
    f_gl :=
      Crn = 0 ∧ Seq ∈ {2,7,9}
    f_gr :=
        (Crn = 2 ∧ Seq ∈ {15,16})
      ∨ (Crn = 5 ∧ Seq ∈ {5,6})
    f_cl := Crn = 2 ∧ Seq = 3
    f_op := Crn = 7
  }
}
```

L'élément séquenceur de la figure 5.2 constitue un *automate de langage légal* classique (contexte usuel de SCT). Pour pouvoir l'utiliser dans un modèle de STS sans structure verticale (comme préconisé par cette méthode de conception) il faut le *fermer* sur l'ensemble des événements incontrôlables qu'il utilise et partage avec les autres dispositifs du sous-système. On a donc les annotations (4) à (11) sur la figure qui correspondent à des requis de contrôle *standards* dans la définition de la composante. Les requis correspondants peuvent être déduits par la méthode exposée au chapitre 4 de [39]. Aussi, le séquenceur définit *explicitement* l'usage de certains événements contrôlables dans une séquence spécifique. Ce sont les contraintes numérotées (12) à (14) dans la spécification de contrôle. On y retrouve l'événement synthétique du sous-système. Finalement les contraintes numérotées (1) à (3) sont des contraintes qui spécifient principalement des conditions de sécurité par exclusion mutuelle entre les états que peuvent adopter les dispositifs physiques. Elles sont dites *implicites* car elles ne requièrent pas le support d'un élément de mémoire pour leur formulation, l'information sur les états des dispositifs physiques suffit (voir la section 4.2.3.2).

La composante locale *DStn* encapsule un modèle brut de 11 016 états si on compte le séquenceur comme une composante du sous-système. Il nécessite la formulation de 22 requis de contrôle auxquels s'ajoutent les 10 requis de contrôle réutilisés par instanciation de composantes réutilisables pour un total de 32 requis de contrôle pour le sous-système.

5.2 Sous-système TStn, la station de test

La station de test a comme fonction de soumettre les pièces, qui lui sont présentées par la station de livraison, à un test de contrôle de qualité concernant l'épaisseur de la pièce. Une pièce répondant aux tolérances quant à l'épaisseur est admise dans un tampon à l'entrée de la station d'usinage. Une pièce hors tolérance est rejetée. Le tampon ne peut contenir qu'une seule pièce au maximum. On trouve en figure 5.4 un diagramme d'assemblage des éléments du problème de contrôle de la station de test avec le détail de son séquenceur (un élément de mémoire) en figure 5.5. La figure 5.6 donne le modèle d'interface $(G,\overline{G})$ de la solution (la composante locale *TStn*).

La station de test est munie d'une plateforme montée sur un vérin monte-charge (*Lift* dans la figure 5.4) lui permettant d'adopter deux positions :

- la position haute où se trouve la sonde d'un micromètre (*MM* dans la figure 5.4) montée sur un vérin ainsi que l'entrée du tampon de préusinage ;
- la position basse où se trouve une chute de rejet pour les pièces non conformes.

La plateforme elle-même est munie d'un vérin d'évacuation (*Evc* dans la figure 5.4) lui

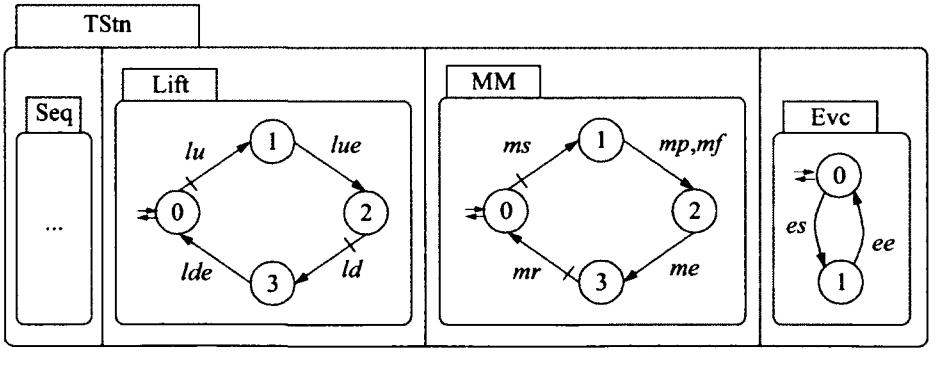


FIGURE 5.4 – Sous-système *TStn*, diagramme d'assemblage

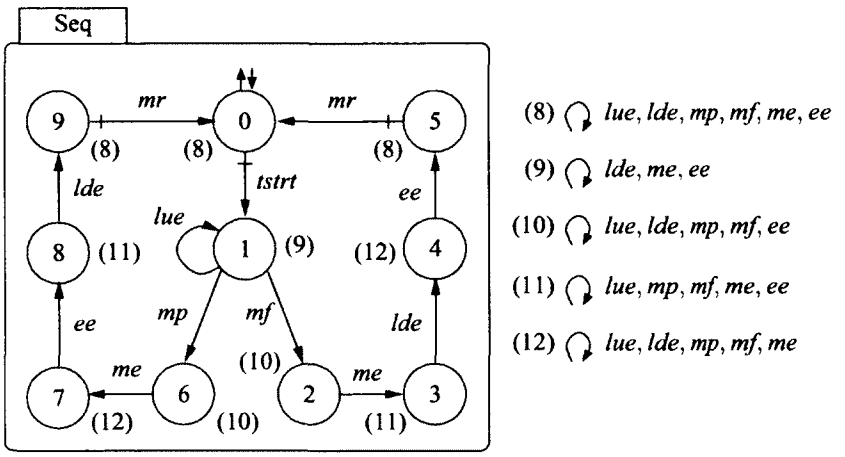


FIGURE 5.5 – Sous-système *TStn*, détails du séquenceur

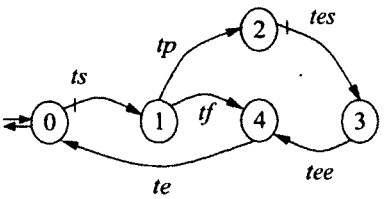
permettant de pousser une pièce hors de la plateforme dans une direction prédéterminée. L'état initial de la station de test (au repos quand la plateforme est vide) se trouve en position basse, le vérin évacuateur hors fonction et le micromètre hors fonction (sonde retirée et micromètre à l'état initial).

Informellement, sur réception d'une pièce à tester, la station doit d'abord effectuer un test de tolérance. Or la sonde du micromètre est située en position haute. Le monte-charge est donc d'abord commandé en position haute puis le micromètre enclenché. Sur l'émission du résultat de test par le micromètre l'une d'entre deux séquences d'actions possibles doit être sélectionnée. Mais dans les deux cas il faut d'abord attendre que la sonde du micromètre se soit désengagée complètement puisqu'il y a une possibilité d'interaction nuisible avec l'évacuateur. Lorsque le test est positif la pièce est admissible à l'usinage. Puisque le monte-charge est déjà en position pour le chargement du tampon de préusinage, l'évacuateur peut tout de suite être enclenché puis le monte-charge redescendu en position basse. Lorsque le test est négatif la pièce doit être rejetée. Or la chute de rejet se situe en position basse. Le monte-charge est donc d'abord commandé en position basse puis l'évacuateur enclenché. Après le traitement complet d'une pièce (chargement au tampon ou rejet) le micromètre est réarmé terminant le cycle complet de la station de test (retour à l'état initial en attente d'une pièce à tester).

Les détails structurels de la composante locale *TStn*, comprenant la spécification for-

TStn : $\overline{G}$

ts	: démarrage de test
tp	: succès
tf	: échec
te	: fin de test
tes	: début de transfert
tee	: fin de transfert



TStn : G

Lift

lu	: lift up
lue	: lift up end
ld	: lift down
lde	: lift down end

MM

ms	: MM start
mp	: MM test pass
mf	: MM test fail
me	: MM test end
mr	: MM reset

Evc

es	: evacuator start
ee	: evacuator end

Synthetic

tstrt	: test station start

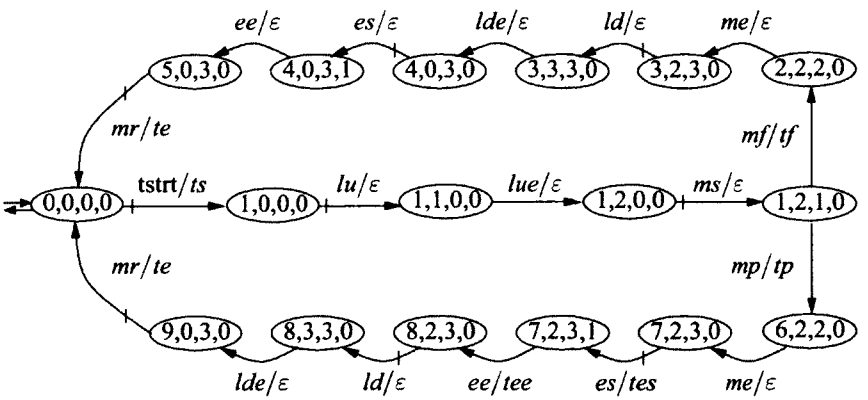


FIGURE 5.6 – Sous-système *TStn* et projection

melle de ses requis de contrôle ainsi que la formulation de la loi de contrôle obtenue par synthèse sur l'ensemble des contraintes, sont donnés dans la spécification de composante qui suit.

```
Component TStn {

  SubSystem TStn {
    State Vector = <Seq, Lift, MM, Evc>

    Seq : MW; /* Sequencer memory 0..9 */
    Lift : Jack211a with {
      Σc := { deploy[lu], retract[ld] }
      Σu := { deployend[lue], retractend[lde] }
    }
    MM : MicroMeter with {
      Σc := { test[ms], reset[mr] }
      Σu := { end[me], pass[mp], fail[mf] }
    }
    Evc : Jack110SS with {
      Σc := { start[es] }
      Σu := { stop[ee] }
    }
    Synthetic := { Σc := { tstrt } }

    CtrlSpec {
    /* Implicit mainly safety */
    /* Lift x MM */
    /*(1) */ Lift ∈ {1,3} ∧ MM ∈ {1,2}
    /*(2) */ Lift = 0 ∧ MM ∈ {1,2}
    /*(3) */ Lift = 1 ∧ MM = 3
    /*(4) */ Lift = 3 ∧ MM = 0
    /* Lift x Evc */
    /*(5) */ Lift ∈ {1,3} ∧ Evc = 1
    /* MM x Evc */
    /*(6) */ MM = ∈ {1,2} ∧ Evc = 1
    /*(7) */ MM = 0 ∧ Evc = 1
    /* Standard wrt Seq, cover safety and add fct */
    /*(8) */ Seq ∈ {0,5,9} ∧
```

```
            ( Lift ∈ {1,3} ∨ MM ∈ {1,2} ∨ Evc = 1 )
/*(9) */ Seq = 1 ∧ ( Lift = 3 ∨ MM = 2 ∨ Evc = 1 )
/*(10)*/ Seq ∈ {2,6} ∧
            ( Lift ∈ {1,3} ∨ MM = 1 ∨ Evc = 1 )
/*(11)*/ Seq ∈ {3,8} ∧
            ( Lift = 1 ∨ MM ∈ {1,2} ∨ Evc = 1 )
/*(12)*/ Seq ∈ {4,7} ∧
            ( Lift ∈ {1,3} ∨ MM ∈ {1,2} )
/*(13)    tstrt <= Seq = 0 */
}

f_tstrt :=
  Seq = 0
f_lu :=
  Lift = 0 ∧ Seq = 1
f_ld :=
  Lift = 2 ∧ Seq = ∈ {3,8}
f_ms :=
  MM = 0 ∧ Lift = 2
f_mr :=
  MM = 3 ∧ Seq ∈ {9,5}
f_es :=
  Evc = 0 ∧ Seq ∈ {4,7}

  }
}
```

Puisque l'activité de la station de test est totalement séquentielle, l'usage d'un *séquenceur* en figure 5.5 (un automate de langage légal) est assez naturelle. Sa structure est d'ailleurs beaucoup plus simple que celle du séquenceur de la station de livraison (en figure 5.2) qui admet un certain niveau de parallélisme dans son action. Les mêmes remarques que pour le séquenceur de la station de livraison sont applicables au séquenceur de la figure 5.5 quant à la façon d'obtenir les requis dit *standards* numérotés (8) à (12). La même numérotation est suivie dans la spécification de contrôle pour la composante locale *TStn*. Encore une fois, le séquenceur contient une contrainte *explicite*, numérotée (13), sur l'événement *synthétique* contrôlable *tstrt* correspondant à une commande de démarrage de la station de test. Le reste des requis de contrôle, numérotés (1) à (7), sont des contraintes

implicites.

- Les contraintes (1), (5) et (6) spécifient que l'action des dispositifs de la station doit être strictement alternée (pas de parallélisme – sécurité).
- La contrainte (2) interdit l'entrée en action du micromètre lorsque le monte-charge est en position basse.
- La contrainte (3) interdit au monte-charge de remonter lorsqu'un test a déjà été fait et la contrainte (4) de redescendre avant que le test ne soit réalisé.
- La contrainte (7) interdit que l'évacuateur soit enclenché sans connaître d'abord le résultat du test.

La composante locale *TStn* encapsule un modèle brut de 69 120 états si on compte le séquenceur comme une composante du sous-système. Il nécessite la formulation de 22 requis de contrôle auxquels s'ajoutent les 14 requis de contrôle réutilisés par instanciation de composantes réutilisables pour total de 36 requis de contrôle pour le sous-système.

5.3 Sous-système PStn, la station d'usinage

La station d'usinage est plus complexe que les deux autres. Il s'agit d'un sous-système construit sur le modèle d'une ligne de montage. Elle comprend une table d'usinage (un convoyeur circulaire) le long de laquelle sont disposées des positions d'usinage effectuant des transformations graduelles sur des pièces à traiter avec un contrôle de qualité à la fin. Ici le sous-système est réduit à sa plus simple expression et ne contient que quatre positions :

- une position d'entrée où les pièces sont admises à l'usinage ;
- une position de perçage ;
- une position pour un test de contrôle de qualité du perçage ;
- une position de sortie.

La figure 5.7 donne un diagramme d'assemblage du sous-système. Un vérin d'obstruction (*Buf* en figure 5.7), disposé à la sortie du tampon de préusinage, est utilisé pour contrôler l'admission des pièces à la position d'entrée. Un mécanisme d'entraînement du convoyeur (*Cvr* en figure 5.7) permet de faire progresser la table d'usinage, une position à la fois, de la position d'entrée vers la position de sortie. Une perceuse (*Drill* en figure 5.7), disposée juste après la position d'entrée, permet d'effectuer une perforation dans la pièce. Une sonde

de profondeur (*Tstr* en figure 5.7) disposée juste après la position de la perceuse permet d'évaluer si la perforation effectuée par la perceuse est assez profonde. Enfin, à la position de sortie, une grue de transfert (*Crn* en figure 5.7) est responsable d'extraire la pièce traitée et de la transférer vers la sortie du système ou de la rejeter, dépendamment du résultat du contrôle de qualité. La figure 5.8 et la figure 5.9 donnent des détails complémentaires sur les éléments de mémoire du sous-système. Enfin, la figure 5.10 donne le modèle d'interface $(G,\overline{G})$ du sous-système (la composante locale *PStn*).

Informellement, la station d'usinage se trouve à l'état initial (au repos) lorsque la table d'usinage est vide et que tous les dispositifs de la ligne de montage (vérin d'obstruction, perceuse, sonde et grue de transfert) sont au repos (hors fonction). À l'admission d'une pièce à l'usinage, cette pièce doit être percée puis sa perforation testée et, dépendamment du résultat de ce test, la pièce doit ensuite être rejetée ou transférée à la sortie du système. Plusieurs pièces peuvent être en traitement simultanément. Les dispositifs d'usinage ne doivent pas entrer en fonction inutilement (sur une position vide par exemple) et ne doivent entrer en fonction qu'une seule fois sur une même pièce.

Les détails structurels de la composante locale *PStn*, comprenant la spécification formelle de ses requis de contrôle ainsi que la formulation de la loi de contrôle obtenue par synthèse sur l'ensemble des contraintes, sont donnés dans la spécification de composante qui suit.

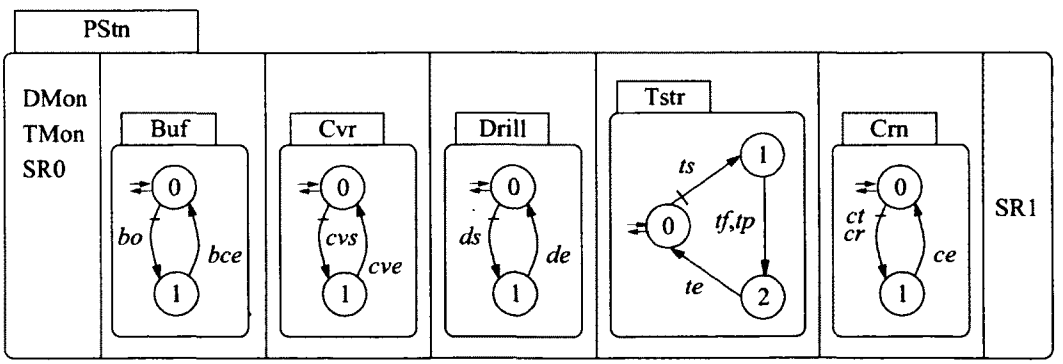


FIGURE 5.7 – Sous-système *PStn*, diagramme d'assemblage

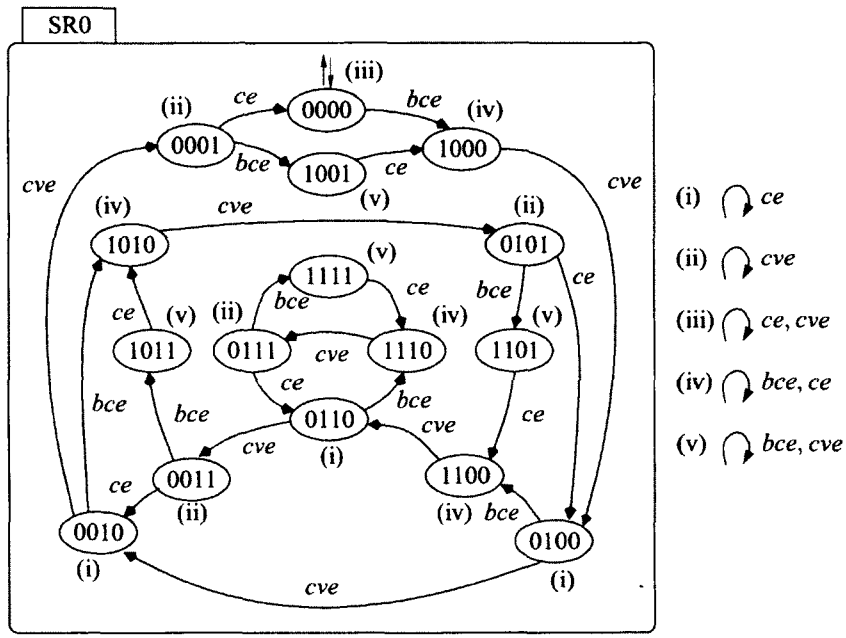


FIGURE 5.8 – Sous-système *PStn*, détails de SR0

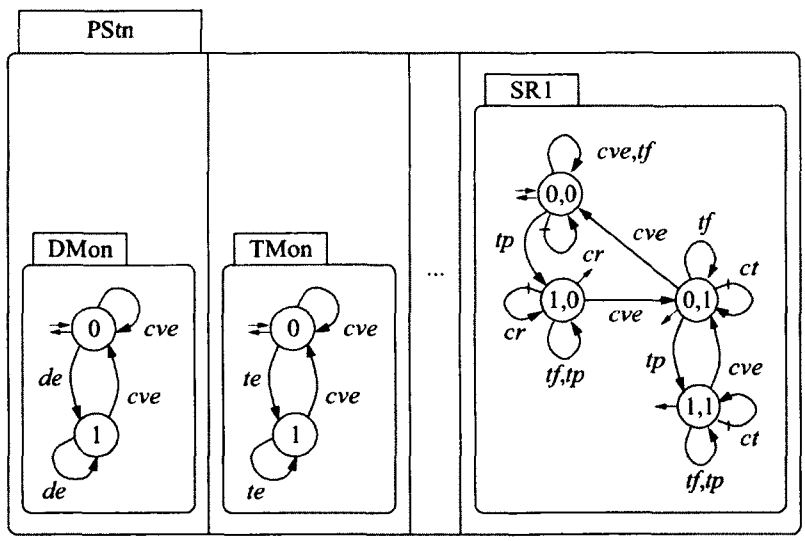
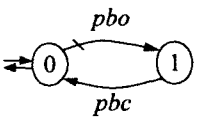


FIGURE 5.9 – Sous-système *PStn*, détails de DMon, TMon et SR1

PStn : $\overline{G}$
pbo : ouverture du tampon
pbc : fermeture du tampon



PStn : G

Buf
bo : buffer open
bce : buffer close end

Cvr
cvs : conveyor step
cve : conveyor step end

Drill
ds : drill start
de : drill end

Tstr
ts : tester start
te : tester end
tp : test pass
tf : test fail

Crn
ct : crane transfer
cr : crane reject
ce : crane pick-up done

Le système sous contrôle comporte 526 états dont 2 sont terminaux, et 1 328 transitions.

La machine de Mealy correspondante projette toutes les transitions étiquetées *bo* sur l'événement *pbo* (168 transitions) et toutes les transitions étiquetées *bce* sur l'événement *pbc* (168 transitions).

FIGURE 5.10 – Sous-système *PStn*, détails et projection

```
Component PStn {
  SubSystem PStn {
    State Vector = <DMon, TMon, SR0, Buf, Cvr, Drill, Tstr,
                    Crn, SR1>
    DMon : MW; /* Drill done  Monitor 0..1 */
    TMon : MW; /* Tester done Monitor 0..1 */
    SR0<M0,M1,M2,M3> : MW; /* Work Table state 0,..,15 */
    SR1<M4,M5> : MW;       /* Test Result double buffer */

    Buf : Jack100SS with {
      Σc := { start[bo] }
      Σu := { stop[bce] }
    }
    Cvr : TSCvrDrv with {
      Σc := { step[cvs] }
      Σu := { end[cve] }
    }
    Drill : Drill with {
      Σc := { start[ds] }
      Σu := { end[de] }
    }
    Tstr : PSTester with {
      Σc := { test[ts] }
      Σu := { pass[tp], fail[tf], end[te] }
    }
    Crn : FHHSCrane with {
      Σc := { trsf[ct], rjct[cr] }
      Σu := { pue[ce] }
    }

    CtrlSpec {
    /* Implicit (safety) */
    /*(1) */ Cvr = 1 ∧ Buf = 1
    /*(2) */ Cvr = 1 ∧ Drill = 1
    /*(3) */ Cvr = 1 ∧ Tstr ∈ {1,2}
    /*(4) */ Cvr = 1 ∧ Crn = 1
    /* Standard wrt SR0 (sequencing) */
    /*(5) */ Buf = 1 ∧
```

```
            SR0 ∈ { 1000, 1001, 1010, 1011, 1100, 1101,
                    1110, 1111 }
   /*(6) */ Crn = 1 ∧
            SR0 ∈ { 0000, 0010, 0100, 0110, 1000, 1010,
                    1100, 1110 }
   /*(7) */ Cvr = 1 ∧
            SR0 ∈ { 0000, 0001, 0011, 0101, 0111, 1001,
                    1011, 1101, 1111 }
   /* Implicit wrt SR0 (sequencing) */
   /*(8) */ Drill = 1 ∧
            SR0 ∈ { 0000, 0001, 0010, 0011, 1000, 1001,
                    1010, 1011 }
   /*(9) */ Tstr = 1 ∧
            SR0 ∈ { 0000, 0001, 0100, 0101, 1000, 1001,
                    1100, 1101 }
   /* Standard wrt DMon and TMon (sequencing) */
   /*(10)*/ DMon = 1 ∧ Drill = 1
   /*(11)*/ TMon = 1 ∧ Tstr = 2
   /* Implicit wrt DMon, TMon, Cvr and SR0 (sequencing) */
   /*(12)*/ DMon = 0
            ∧ SR0 ∈ { 0100, 0101, 0110, 0111, 1100, 1101,
                      1110, 1111 }
            ∧ Cvr = 1
   /*(13)*/ TMon = 0 ∧
            ∧ SR0 ∈ { 0010, 0011, 0110, 0111, 1010, 1011,
                      1110, 1111 }
            ∧ Cvr = 1
   /* Explicit in SR1 (sequencing) */
   /*(14)    cr <= SR1 ∈ { 00, 10 } */
   /*(15)    ct <= SR1 ∈ { 01, 11 } */
   }

/* Control Law : */
f_bo :=
  Buf = 0
  ∧ Cvr = 0
  ∧ SR0 ∈ { 0000, 0001, 0010, 0011, 0100, 0101, 0110,
            0111 }
```

```
  f_cvs :=
    Cvr = 0
    ∧ Buf = 0
    ∧ Drill = 0
    ∧ Tstr = 0
    ∧ Crn = 0
    ∧ (   SR0 = 1000
        ∨ ( SR0 ∈ { 0100, 1100 } ∧ DMon = 1 )
        ∨ ( SR0 ∈ { 0010, 1010 } ∧ TMon = 1 )
        ∨ ( SR0 ∈ { 0110, 1110 } ∧ DMon = 1 ∧ TMon = 1 )
      )
  f_ds :=
    Drill = 0
    ∧ Cvr = 0
    ∧ DMon = 0
    ∧ SR0 ∈ { 0100, 0101, 0110, 0111, 1100, 1101, 1110,
              1111 }
  f_ts :=
    Tstr = 0
    ∧ Cvr = 0
    ∧ TMon = 0
    ∧ SR0 ∈ { 0010, 0011, 0110, 0111, 1010, 1011, 1110,
              1111 }
  f_cr :=
    Crn = 0
    ∧ Cvr = 0
    ∧ SR0 ∈ { 0001, 0011, 0101, 0111, 1001, 1011, 1101,
              1111 }
    ∧ SR1 ∈ { 00, 10 }
  f_ct :=
    Crn = 0
    ∧ Cvr = 0
    ∧ SR0 ∈ { 0001, 0011, 0101, 0111, 1001, 1011, 1101,
              1111 }
    ∧ SR1 ∈ { 01, 11 }
  }
}
```

Il est préférable d'aborder la spécification de contrôle de la station d'usinage par étapes : contraintes de sécurité d'abord, coordination du séquencement global des activités ensuite et finalement contraintes relatives aux particularités des différentes positions d'usinage.

Les contraintes numérotées (1) à (4) dans la spécification concernent la sécurité des dispositifs. Elles stipulent simplement que le convoyeur ne peut être en fonction en même temps que les dispositifs d'usinage, contraintes (2) et (3), que le vérin d'obstruction du tampon, contrainte (1), ou pendant que la grue est en train d'extraire une pièce de la position de sortie, contrainte (4). Ces contraintes sont *implicites* car l'état des dispositifs impliqués suffit comme support à leur formulation.

Les contraintes numérotées (5) à (7) concernent la coordination de l'activité globale de la station. Pour spécifier ces contraintes il est nécessaire de modéliser l'état des différentes positions du convoyeur par un élément de mémoire ; le registre à décalage SR0 dont le schéma est donné en figure 5.8. Dans ce schéma les états sont dénotés par des chaînes de bits de la forme «*entrée, perceuse, sonde, sortie*» correspondant aux positions sur la table d'usinage. Par exemple, l'état identifié «0101» correspond à une configuration de la table d'usinage où la position de la perceuse et la position de sortie sont occupées par une pièce et les autres positions sont vides. Cette forme de notation est jugée plus claire pour les registres à décalage. Puisqu'il s'agit d'un élément de mémoire il faut *fermer* son modèle sur l'ensemble des événements incontrôlables qu'il utilise (annotations (i) à (v) dans la figure 5.8). Les contraintes qui en résultent (des contraintes *standards*) sont déduites par la procédure donnée dans [39] au chapitre 4. On ne peut admettre une pièce à l'usinage que si la position d'entrée est vide ; contrainte (5). La grue de transfert ne peut entrer en fonction que si une pièce est présente à la position de sortie ; contrainte (6). Le convoyeur ne peut entrer en fonction que si la position de sortie est vide (elle a été vidée par l'activité de la grue de transfert) à condition que la table d'usinage ne soit pas vide (configuration «0000»).

Il faut spécifier quelques contraintes pour éviter que la perceuse et la sonde entrent en fonction lorsque les positions correspondantes de la table d'usinage sont vides. Les états des dispositifs impliqués associés à l'état de la table d'usinage (le registre à décalage SR0) suffisent pour supporter la spécification de ces contraintes. Les contraintes (8), pour la perceuse, et (9), pour la sonde, sont donc des contraintes *implicites* empêchant ces dispositifs

d'entrer en fonction sur une position correspondante vide de la table d'usinage.

L'ensemble des contraintes mises en place jusqu'à présent définissent les requis de sécurité et la séquence générale des opérations de la station. Mais avec ces requis la perceuse et la sonde de contrôle de qualité peuvent toujours entrer en fonction 0, 1 ou plusieurs fois sur une même pièce lorsqu'elle se présente à leur position d'usinage respective. Aussi, bien que la grue de transfert entre en opération au moment voulu, rien ne lui permet de sélectionner l'action appropriée à la situation (rejet ou transfert). Pour empêcher la perceuse et la sonde de fonctionner plus d'une fois pour la même pièce et permettre la sélection de l'opération appropriée de la grue il faut ajouter les éléments mémoire illustrés à la figure 5.9 : les moniteurs d'opération *DMon*, *TMon* et le registre à décalage SR1.

Les contraintes (10) et (11) de la spécification de contrôle sont, respectivement, des requis *standards* par rapport aux moniteurs *DMon* et *TMon*. Ils empêchent la perceuse (10) et la sonde (11) d'entrer plus d'une fois en fonction sur une même pièce. Il faut noter que l'événement *cve* est utilisé pour remettre les deux moniteurs à zéro à chaque activation du convoyeur mais que les requis (10) et (11) ne stipulent aucune contrainte sur l'activité du convoyeur (la transition en boucle sur *cve* est légale dans ce contexte). Pour *obliger* la perceuse et la sonde à entrer en fonction lorsqu'une pièce se présente à leur position d'usinage respective, il faut aussi spécifier les contraintes *implicites* (12) et (13). L'information d'état de la table d'usinage (SR0 en figure 5.8) alliée à celles du moniteur (*DMon* en figure 5.9) et du convoyeur suffisent pour stipuler une contrainte *implicite* (12), qui a pour effet d'empêcher le convoyeur d'entrer en fonction avant que la perceuse n'ait eu le temps de faire son travail. Un raisonnement analogue peut être fait pour la contrainte (13) relativement à la sonde de contrôle de qualité.

Finalement, le registre à décalage SR1, dont le modèle est illustré à la figure 5.9, sert à mémoriser le résultat du test de la sonde de contrôle pour ensuite le *transporter* en synchronisme avec la pièce correspondante. Cette manoeuvre est nécessaire car le résultat du test à la sonde de contrôle de qualité doit servir *subséquemment* (après un cycle du convoyeur) à la grue de transfert pour sélectionner l'action appropriée lors de son entrée en fonction (rejet ou transfert). Sans l'usage d'un *double tampon* comme SR1, il y a un risque que la sonde de contrôle de qualité *détruise* ce résultat de test *avant* qu'il n'ait été utilisé. Les requis (14) et (15) sont *explicites* dans le modèle de SR1, il s'agit des transitions en boucle

sur les événements contrôlables *ct* et *cr* partagés avec la grue de transfert. Il faut noter que, mis à part les requis (14) et (15), SR1 n'impose aucune autre contrainte. Le modèle de SR1 est *fermé* sur l'ensemble des événements incontrôlables qu'il partage avec les autres dispositifs et toutes ces transitions sont *légales* dans ce contexte. De plus, SR1 n'impose aucune contrainte de *marquage*, c'est-à-dire que tous ses états sont *terminaux*.

Dans le modèle de la projection de la composante locale *PStn* illustré en figure 5.10 il est impossible de donner une représentation graphique de la projection (la machine de Mealy G) car elle est trop volumineuse. La fonction de projection est cependant très simple car il suffit de projeter toutes les transitions qui correspondent à une activité du vérin d'obstruction du tampon de préusinage à l'entrée de la station (événements *bo* et *bce*).

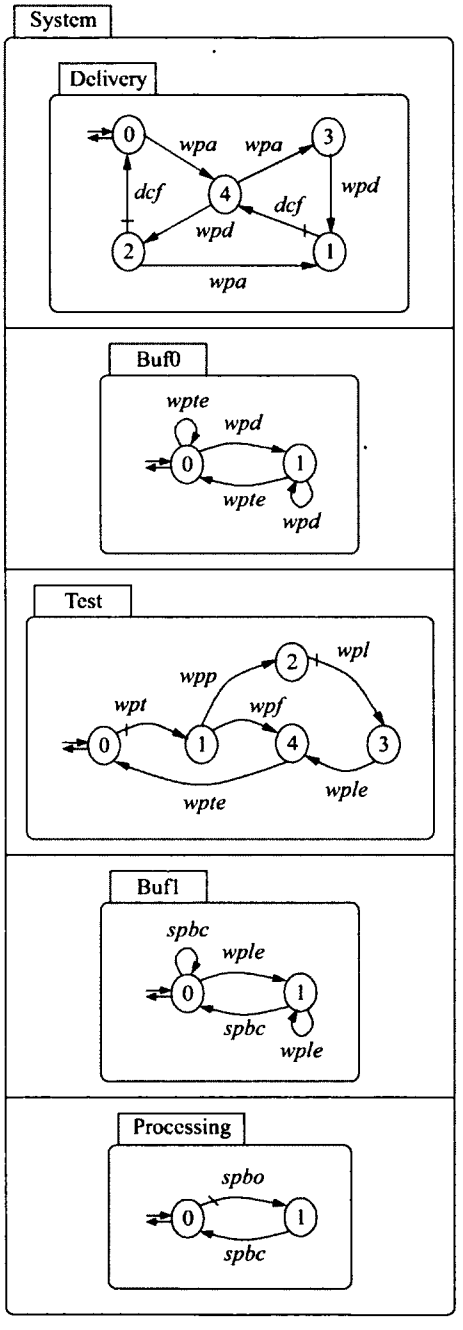
La composante locale *PStn* encapsule un modèle brut de l'ordre de $7{,}13 \times 10^{13}$ états si on compte les éléments de mémoire *DMon*, *TMon*, SR0 et SR1 comme composantes du sous-système. Il nécessite la formulation de 15 requis de contrôle auxquels s'ajoutent les 79 requis de contrôle réutilisés par instanciation de composantes réutilisables pour un total de 94 requis de contrôle pour le sous-système.

5.4 Assemblage du système

La figure 5.11 donne un diagramme d'assemblage pour le système complet. Comme on peut le voir le système ne nécessite que l'adjonction de deux éléments de mémoire (deux tampons de capacité d'une pièce) pour qu'il soit possible de coordonner les trois sous-systèmes correspondant à des instances des composantes locales *DStn*, *TStn* et *PStn*. Les requis de contrôle du système (tous standards) sont déduits de la façon usuelle (exposée dans les sections précédentes).

Le système encapsule un modèle brut de l'ordre de $2{,}17 \times 10^{23}$ états. Il nécessite la formulation de 4 requis de contrôle auxquels s'ajoutent les 154 requis de contrôle réutilisés par l'instanciation des composantes locales pour un total de 158 requis de contrôle.

Les détails structurels du système, comprenant la spécification formelle de ses requis de contrôle ainsi que la formulation de la loi de contrôle obtenue par synthèse sur l'ensemble des contraintes, sont donnés dans la spécification de système qui suit.



Delivery
wpa : arrivée d'une pièce
wpd : pièce livrée
dcf : libérer la grue

Test
wpt : démarrage de test
wpp : succès
wpf : échec
wpte fin de test
wpl : début de transfert
wple fin de transfert

Processing
spbo : ouverture du tampon
spbc : fermeture du tampon

FIGURE 5.11 – Diagramme d'assemblage du système

```
System {
  State Vector = <Delivery, Buf0, Test, Buf1, Processing>

    Buf0 : MW; /* Delivery to Test Buffer 0..1 */
    Buf1 : MW; /* Test to Processing Buffer 0..1 */

   Delivery : DStn with {
      Σc := { drst[dcf] }
      Σu := { dwp[wpa], dgle[wpd] }
   }
   Test : TStn with {
      Σc := { ts[wpt], tes[wpl] }
      Σu := { tp[wpp], tf[wpf], te[wpte], tee[wple] }
   }
   Processing : PStn with {
     Σc := { pbo[spbo] }
     Σu := { pbc[spbc] }
   }

    CtrlSpec {
    /* Standard wrt Buf0 */
    /*(1)*/ Buf0 = 0 ∧ Test = 4
    /*(2)*/ Buf0 = 1 ∧ Delivery ∈ {3,4}
    /* Standard wrt Buf1 */
    /*(3)*/ Buf1 = 0 ∧ Processing = 1
    /*(4)*/ Buf1 = 1 ∧ Test = 3
    }
  /* Control Law : */
  ∫dcf :=
    Delivery ∈ {1,2} ∧ Buf0 = 0
  ∫wpt :=
    Test = 0 ∧ Buf0 = 1
  ∫wpl :=
    Test = 2 ∧ Buf1 = 0
  ∫spbo :=
    Processing = 0 ∧ Buf1 = 1
}
```

Conclusion

Le temps est maintenant venu d'établir un bilan des contributions de cette thèse. Il est possible d'affirmer, sans exagérer, que tous les objectifs fixés initialement sont atteints.

Sommaire des contributions

Après une revue détaillée de la documentation de recherche sur la théorie et sur ses variantes (au chapitre 1 et au chapitre 2), il est possible de formuler une métaphore de composantes réutilisables, moyennant l'ajout de certains résultats mineurs complémentaires à la théorie du contrôle hiérarchique de Wong et Wonham [55, 58, 59] (au chapitre 3). Cette métaphore de composantes réutilisables fournit les bases d'une méthode d'abstraction et d'encapsulation des requis de contrôle. Dans le cas des composantes élémentaires on obtient principalement une façon de réutiliser (et éventuellement rentabiliser) le travail de modélisation *ad hoc* qui est nécessaire pour passer du domaine informel au domaine formel. Dans le cas des composantes mixtes on obtient une encapsulation et une réutilisation des requis de contrôle de portée locale simplifiant du coup tant la structure de transition de ces composantes que la formulation des requis de contrôle pour les sous-systèmes qui les utilisent (voir les exemples du chapitre 4).

En conjonction avec la métaphore de composantes réutilisables il est aussi possible de formuler les détails des mécanismes d'instanciation de ces composantes et de leur composition en sous-systèmes pour la formulation de problèmes de contrôle (au chapitre 4). La méthode proposée de construction des sous-systèmes (sous forme de STS sans structure verticale) offre à son tour l'avantage de pouvoir utiliser des prédicats pour l'expression des requis de contrôle (voir chapitre 4, section 4.1.1). Dans l'étude de cas (au chapitre 5), il

devient vite évident que cette forme d'expression des requis de contrôle est supérieure à l'usage d'AFD de contrainte en termes de clarté d'expression. Les requis formels peuvent, entre autres, être mis en correspondance avec les requis informels qui les motivent. Aussi cette forme (par usage de prédicats) est mieux connue dans la pratique et certainement plus abordable par des non-spécialistes que celle des AFD de contrainte.

La forme de ces sous-systèmes et l'expression de leurs requis de contrôle par des prédicats sont parfaitement compatibles avec les procédures de synthèse de SCT (voir au chapitre 4). Il est d'ailleurs possible d'utiliser autant les procédures de synthèse *classiques* (qui fonctionnent par exploration exhaustive de l'espace des états) que les procédures spécialisées formulées par Ma et Wonham dans [39] (avec usage de BDD). L'usage des procédures de Ma et Wonham est cependant plus adéquate puisque ces procédures produisent directement une loi de contrôle sous forme d'une SFBC. Quoi qu'il en soit, étant donné la forme de modélisation des problèmes de contrôle préconisée (des STS sans structure verticale), le résultat peut toujours être réduit à un ensemble de prédicats de contrôle (une SFBC, un de nos objectifs initiaux). Ce type de résultat de synthèse est nécessaire pour une implémentation efficace de la loi de contrôle.

Enfin, la définition d'une *dynamique abstraite* (au chapitre 4) pour les composantes réutilisables procure l'instrumentation nécessaire pour la formulation de schémas de génération de code compatibles avec une variété de cibles. En donnant une forme précise aux *heuristiques de génération de code*, du code d'intégration (*glue code*) peut être inséré aux endroits appropriés par le générateur de code lui-même. Il est clair que l'ensemble des cibles possibles pour la génération de code ne se limite pas à des cibles PLC. Avec la définition de la dynamique abstraite le contrôle peut être envisagé sur un ordinateur plus conventionnel (par exemple, *PC-based control* dans le jargon de la pratique).

La *dynamique abstraite* est basée sur l'*appel de procédures* et une lecture inattentive peut porter à croire qu'une cible PLC doit être dotée d'un mécanisme de transfert de flot de contrôle adéquat pour pouvoir supporter cette dynamique, mais il n'en est rien. Dans la *dynamique abstraite* on peut remarquer que le flot des appels de procédure passe toujours des composantes les plus concrètes aux plus abstraites, formant un graphe orienté acyclique qui s'arrête au niveau du système. Avec cette forme de structure d'appel de procédure, les diverses fonctions appelées peuvent être *incluses* au point d'appel. Il s'agit d'une technique

bien connue en génération de code et aussi en programmation, la technique s'appelle *in-lining* dans son appellation anglophone. Bref, cette *dynamique abstraite* est adéquate pour la génération de code avec une cible PLC (ce qui est l'un de nos objectifs initiaux).

Critique des résultats obtenus

Tous les objectifs initiaux sont atteints et l'ensemble de l'instrumentation proposée est adéquat pour produire un environnement de conception de contrôleurs SCT intégré. Ceci dit, l'instrumentation conçue ne constitue pas une panacée puisque certains problèmes subsistent.

Bien qu'il soit raisonnable de sacrifier le critère d'*optimalité* de la synthèse pour un mécanisme d'encapsulation procurant la réutilisation de composantes, certaines situations demeurent problématiques. Dans l'étude de cas (au chapitre 5), le tampon servant à transporter le résultat du contrôle de qualité de la station d'usinage en est un exemple. Ce type de mémoire constitue un *attribut* d'un élément usiné qu'il faut modéliser indépendamment. Ce type de problème est endémique à SCT plutôt qu'à la méthode proposée mais il demeure problématique.

Concernant la méthode elle-même, la division en composantes et en sous-systèmes *force* parfois le concepteur à certaines gymnastiques conceptuelles (par exemple la station de livraison qui n'a pas de capteur à la sortie dans l'étude de cas). Mais, somme toute, les avantages de la réutilisation pour éviter la modélisation *ad hoc* répétée de dispositifs, l'expression des requis par prédicats et la modularisation des requis de contrôle semblent compenser adéquatement.

L'instrumentation logicielle à procurer demeure assez lourde : les algorithmes de synthèse et de Wong en plus d'une bonne interface graphique et d'un générateur de code. Mais puisque la méthode de conception préconisée intègre toutes les phases de la conception, de la modélisation à la génération de code, elle représente un plan de réalisation assez clair avec un résultat bien défini et utilisable. Ce qui semble manquer dans les quelques réalisations actuelles est précisément un objectif quant à la façon d'utiliser un environnement de conception de contrôleurs SCT.

Directions de recherches potentielles

Il reste bien sûr à réaliser un environnement intégré de conception de contrôleurs SCT par composantes réutilisables le long des lignes directrices préconisées ici. Mais l'ensemble des investigations réalisées pour cette thèse ouvrent aussi certaines questions théoriques intéressantes.

Il est inévitable de remarquer que le genre de composantes réutilisables préconisées dans la thèse est éminemment utilisable dans les autres variantes hiérarchiques de la théorie. Par exemple, une projection qui est un *observateur* peut aisément posséder la structure d'une *interface en paires commandes/réponses* (*command pair interfaces* dans HISC [35, 38]). Le modèle abstrait de la plupart des vérins donné en annexe D y correspond, si on les transforme en *observateurs* naturels comme dans [14]. L'inverse n'est pas nécessairement vrai. Le critère d'observateur est très strict. Les *interfaces en paires commandes/réponses* représentent un critère plus faible permettant une meilleure agrégation de l'espace des états. Il y a ainsi lieu de s'interroger à savoir s'il ne serait pas possible de reformuler certains résultats de la théorie de Wong et Wonham pour permettre l'usage de projections ayant des propriétés similaires à celles des *interfaces en paires commandes/réponses* pour obtenir une agrégation de l'espace des états plus importante dans les composantes réutilisables. Autrement dit : peut-on affaiblir le requis de la propriété d'observateur en contrôle hiérarchique ? Le travail de Leduc *et al* semble indiquer que oui.

De la même façon, rien n'empêche l'usage des composantes réutilisables préconisées dans un problème de contrôle spécifié par STS tel que dans [39]. Bien sûr l'application des résultats de synthèse obtenus doit tenir compte de l'encapsulation réalisée par l'usage des composantes réutilisables. L'encapsulation, on le rappelle, est ici *opaque* (voir au chapitre 3) alors que l'encapsulation offerte par les STS est *ouverte*. Il y a peut-être là, dans la relation entre encapsulation opaque et encapsulation ouverte, une extension possible et intéressante de l'outil de modélisation qu'offrent les STS. Bien que la modélisation du parallélisme chez les STS soit très élégante, on se rappelle qu'il interdit, à toute fin pratique, la modélisation d'une transition entre deux noeuds conjonctifs de même structure. Par exemple, le modèle du registre à décalage SR0 de la station d'usinage (dans l'étude de cas au chapitre 5) ne peut pas utiliser un noeud disjonctif formé d'une structure de transi-

tion entre noeuds conjonctifs, ce qui serait pourtant le modèle le plus expressif.

Aussi, dans [59], le schéma présenté par Wong et Wonham, appelé la *coordination*, peut, peut-être, être utilisé pour l'analyse de situations conflictuelles entre des composantes qui *partagent* certains de leurs événements. Ceci peut ouvrir la porte à des composantes qui partagent certains de leurs événements. Un tel schéma est plus flexible et vaut la peine d'être investigué. Quoi qu'il en soit, la *coordination* présente des similitudes avec les concepts étudiés dans cette thèse qui méritent d'être clarifiés pour en comprendre les implications.

Épilogue

En terminant on peut observer que les résultats de cette thèse constituent probablement le genre de réponse adéquate à l'article de Ekberg et Krogh [13]. À peu de chose près, la métaphore de composantes réutilisables utilisée ici s'assimile aux *schémas* de machines d'états (*State Machine Templates*) préconisés dans cet article. Puisque les prédicats de contrôle, qui forment ici le résultat de synthèse, peuvent être réduits à des conditions de garde sur des transitions contrôlables (voir au chapitre 4), il semble bien que ce modèle soit très similaire au leur.

Annexe A

Résultats sur le contrôle non bloquant

Ce qui suit est une spécialisation des résultats de Wong et Wonham [58] concernant le contrôle non bloquant dans les hypothèses du théorème 2 (p. 254). Notamment on cherche à prouver que ces résultats tiennent toujours lorsque $\mathcal{C}_{hi}(M) \subseteq \theta(\mathcal{C}_{lo}(L))$ (consistance hiérarchique restreinte) pourvu que la spécification soumise à l'agent soit $E \in \mathcal{C}_{hi}(M)$, c'est-à-dire que la spécification est contrôlable. Le but est d'en arriver au résultat du théorème 6 qui résume tous les résultats à partir de la proposition 15.

Notez que dans ces conditions si on dispose d'un opérateur de synthèse pour l'abstraction, par exemple si M est muni d'une technologie de contrôle standard auquel cas $\kappa_M(E) = \sup\mathcal{C}_M(E)$, on peut toujours soumettre à l'agent le résultat de synthèse de l'abstraction comme spécification ; la spécification est toujours contrôlable. Dans ce cas, la consistance hiérarchique restreinte suffit, bien que le nombre d'options de contrôle disponibles dans $\mathcal{C}_{hi}$ puisse être plus limité que le nombre d'options de contrôle disponibles dans $\mathcal{C}_{lo}$.

Rappelons que pour le développement des résultats sur la synthèse non bloquante dans [58] on doit imposer la restriction suivante sur $\mathcal{C}_{lo}$,

$$(\forall K \in \mathcal{P}(H))\ \overline{K} \in \mathcal{C}_{lo}(H) \implies K \in \mathcal{C}_{lo}(H) \tag{A.1}$$

c'est-à-dire que la contrôlabilité d'un sous-langage est déterminée par la contrôlabilité de sa fermeture préfixée. Une structure de contrôle standard possède cette propriété ([55],

section 3.1).

Rappelons aussi certains résultats sur la propriété d'observateur. Pour Σ et T deux alphabets disjoints, $L \subseteq \Sigma^*$ fermé, $M \subseteq T^*$ et $\theta : L \to M$ une projection causale :

$$\theta \textit{ est un observateur} \iff \theta^{-1} \circ pre_M = pre_L \circ \theta^{-1}$$

où pre_L et pre_M sont les opérateurs de fermeture préfixée sur L et M respectivement ([58], théorème 3). Et,

$$\begin{aligned} & \theta^{-1} \circ pre_M = pre_L \circ \theta^{-1} \\ \iff \quad & (\forall\, s \in L)(\forall\, t \in T^+)\ \theta(s)t \in M \implies (\exists\, u \in \Sigma^+)\ su \in L \textit{ et } \theta(su) = \theta(s)t \end{aligned}$$

([58], proposition 7), dont la formulation suivante

$$\begin{aligned} & \theta^{-1} \circ pre_M = pre_L \circ \theta^{-1} \\ \iff \quad & (\forall\, s \in L)(\forall\, \tau \in T)\ \theta(s)\tau \in M \implies (\exists\, u \in \Sigma^+)\ su \in L \textit{ et } \theta(su) = \theta(s)\tau \end{aligned}$$

est souvent plus commode.

Rappelons finalement l'énoncé du théorème 2 concernant la consistance hiérarchique restreinte.

Théorème 2 ([58]).
Soit C_{lo} et C_{hi} deux structures de contrôle respectivement sur L et M, et soit $N \in \mathcal{F}_M$ avec $H := \theta^{-1}(N) \in \mathcal{F}_L$. Alors

$$C_{hi}(N) \subseteq \theta(C_{lo}(H)) \iff \theta \circ \kappa_H \circ \theta^{-1}|_{C_{hi}(N)} = id_{C_{hi}(N)}$$

où id_X est l'identité sur X.

On aura donc toujours $\theta \circ \kappa_L \circ \theta^{-1}(\kappa_M(E)) = \kappa_M(E)$ pour tout $E \subseteq M$ en présence de consistance hiérarchique restreinte. Aussi, κ est idempotent, κ est monotone quant à l'inclusion d'ensembles, et si $N \in C_{hi}(M)$, alors $N = \kappa_M(N)$.

Adaptation de la proposition 15 ([58]).

$$(\forall E \subseteq M)\, \kappa_L \circ \theta^{-1}(\kappa_M(E)) \textit{ non bloquant} \implies \kappa_M(E) \textit{ non bloquant}$$

Démonstration. Posons $E \subseteq M$ et supposons que $\kappa_L \circ \theta^{-1}(\kappa_M(E))$ soit non bloquant. Alors

$$\overline{\overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))} \cap L_m} = \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}$$

et on doit démontrer que $\overline{\overline{\kappa_M(E)} \cap M_m} = \overline{\kappa_M(E)}$.

($\subseteq$) Puisque $\overline{\kappa_M(E)} \cap M_m \subseteq \overline{\kappa_M(E)}$ alors $\overline{\overline{\kappa_M(E)} \cap M_m} \subseteq \overline{\kappa_M(E)}$.
($\supseteq$) Soit $t \in \overline{\kappa_M(E)}$. Alors, puisque $\mathcal{C}_{hi}(M) \subseteq \theta(\mathcal{C}_{lo}(L))$ et θ est causale, on a

$$\begin{aligned} t \in \overline{\kappa_M(E)} &= \overline{\theta \circ \kappa_L \circ \theta^{-1}(\kappa_M(E))} = \overline{\theta(\kappa_L \circ \theta^{-1}(\kappa_M(E)))} \\ &= \theta(\overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}). \end{aligned}$$

Ainsi il existe $u \in \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}$ tel que $\theta(u) = t$. Or, pour u ainsi fixée, puisque $\kappa_L \circ \theta^{-1}(\kappa_M(E))$ est non bloquant par hypothèse, on a

$$\begin{aligned} & u \in \overline{\overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))} \cap L_m} \\ \implies & (\exists\, s \mid s \in \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))} \cap L_m : u \leq s) \\ \implies & (\exists\, s \mid s \in L_m : u \leq s \wedge s \in \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}) \\ \implies & (\exists\, s \mid s \in L_m : \theta(u) \leq \theta(s) \wedge\ \theta(s) \in \theta(\overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))})), \text{ car } \theta \text{ est causale} \\ \implies & (\exists\, s \mid s \in L_m : t \leq \theta(s) \wedge \theta(s) \in \overline{\theta(\kappa_L \circ \theta^{-1}(\kappa_M(E)))}),\ \theta \text{ causale et } \theta(u) = t \\ \implies & (\exists\, s \mid s \in L_m : t \leq \theta(s) \wedge \theta(s) \in \overline{\theta \circ \kappa_L \circ \theta^{-1}(\kappa_M(E))}) \\ \implies & (\exists\, s \mid s \in L_m : t \leq \theta(s) \wedge \theta(s) \in \overline{\kappa_M(E)}),\ \text{car } \mathcal{C}_{hi}(M) \subseteq \theta(\mathcal{C}_{lo}(L)) \\ \implies & (\exists\, s \mid s \in L_m : t \leq \theta(s) \wedge \theta(s) \in \overline{\kappa_M(E)} \cap M_m),\ \text{car } \theta(s) \in \theta(L_m) = M_m \\ \implies & t \in \overline{\overline{\kappa_M(E)} \cap M_m}. \end{aligned}$$

Sachant que t est arbitraire dans $\overline{\kappa_M(E)}$ on a donc $\overline{\overline{\kappa_M(E)} \cap M_m} \supseteq \overline{\kappa_M(E)}$.

D'où on peut conclure que $\overline{\overline{\kappa_M(E)} \cap M_m} = \overline{\kappa_M(E)}$. □

Pour le prochain théorème on considère la notion suivante. On dit que la paire (L, θ) est *fortement observable* si $\theta|_K$, c'est-à-dire θ dont le domaine est restreint à K, est dotée de la propriété d'observateur pour tout $K \in \overline{\mathcal{C}}_{lo}(L)$.

Adaptation du théorème 4 ([58]).
En supposant

$\theta^{-1}(M_m) = L_m$;

(L, θ) *fortement observable* ;

alors

$$(\forall\, E \in M)\ \kappa_M(E) \textit{ non bloquant} \iff \kappa_L \circ \theta^{-1}(\kappa_M(E)) \textit{ non bloquant}$$

Démonstration. Grâce à la proposition 15, il suffit de montrer ($\Longrightarrow$).
Soit $E \subseteq M$. Supposons que $\kappa_M(E)$ soit non bloquant. Alors

$$\overline{\overline{\kappa_M(E)} \cap M_m} = \overline{\kappa_M(E)}. \tag{A.2}$$

On doit prouver $\overline{\overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))} \cap L_m} = \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}$.
($\subseteq$) Puisque $\overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))} \cap L_m \subseteq \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}$, il est clair que

$$\overline{\overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))} \cap L_m} \subseteq \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}.$$

($\supseteq$) Posons $s \in \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}$. Alors

$$\theta(s) \in \theta(\overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}) = \overline{\theta(\kappa_L \circ \theta^{-1}(\kappa_M(E)))} = \overline{\theta \circ \kappa_L \circ \theta^{-1}(\kappa_M(E))}$$

puisque θ est causale. Ainsi $\theta(s) \in \overline{\kappa_M(E)}$ puisque $\mathcal{C}_{hi}(M) \subseteq \theta(\mathcal{C}_{lo}(L))$. Donc il existe $t \in T^*$ pour lequel $\theta(s)t \in \overline{\kappa_M(E)}$ et $\theta(s)t \in M_m$ (éq. (A.2)).
D'autre part, puisque $\kappa_L \circ \theta^{-1}(\kappa_M(E)) \in \mathcal{C}_{lo}(L)$, $K := \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))} \in \overline{\mathcal{C}}_{lo}(L)$.

Posons maintenant $\theta_K := \theta|_K$. Il est clair que

$$\begin{aligned}
\theta_K(K) &= \theta(K) \\
&= \theta(\overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}) \\
&= \overline{\theta(\kappa_L \circ \theta^{-1}(\kappa_M(E)))} \\
&= \overline{\theta \circ \kappa_L \circ \theta^{-1}(\kappa_M(E))} \\
&= \overline{\kappa_M(E)}
\end{aligned}$$

Puisque (L, θ) est fortement observable, θ_K possède la propriété d'observateur. Ainsi, par les propositions 8 et 7 de Wong et Wonham [58], il existe $u \in \Sigma^*$ pour lequel $su \in K$ et $\theta_K(su) = \theta_K(s)t = \theta(s)t = \theta(su)$. On a donc que $\theta_K(su) \in \overline{\kappa_M(E)}$ et $\theta_K(su) \in M_m$. Or $\theta_K(su) = \theta(su) \in M_m$ et donc $su \in \theta^{-1}(M_m) = L_m$. Par conséquent $su \in K \cap L_m = \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))} \cap L_m$, d'où $s \in \overline{\overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))} \cap L_m}$. □

Le but de la proposition 16 et du théorème 5 dans [58] est d'affaiblir l'exigence d'observabilité forte pour la paire (L, θ) de façon à pouvoir fournir éventuellement une condition suffisante garantissant cette propriété sur un ensemble utile de sous-langages. À cette fin on définit

$$\mathcal{X}' := \{K \in \overline{C}_{lo}(L) \mid \kappa_L \circ \theta^{-1} \circ \theta(K) = K\}$$

l'ensemble des sous-langages fermés et contrôlables de L qui sont maximaux par rapport à leur image par θ. Si l'on ne dispose que de la consistance hiérarchique restreinte, il faut restreindre davantage $\mathcal{X}'$. On définit donc

$$\mathcal{X} := \{K \in \overline{C}_{lo}(L) \mid \kappa_L \circ \theta^{-1} \circ \theta(K) = K,\ \theta(K) \in C_{hi}(M)\}$$

pour ne sélectionner dans $\mathcal{X}'$ que les éléments dont l'image est contrôlable dans M comme ensemble de sous-langages intéressants. On dit alors que la paire (L, θ) est *fortement observable par rapport à* $\mathcal{X}$ si $\theta|_K$ est dotée de la propriété d'observateur quel que soit $K \in \mathcal{X}$.

Lemme A.1. *Soit $\mathcal{C}_{hi}(M) \subseteq \theta(\mathcal{C}_{lo}(L))$. Posons $K := \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)})$ et $\theta_K := \theta|_K$. Supposons enfin que θ_K est doté de la propriété d'observateur. Alors*

$$\overline{K \cap \theta^{-1}(\kappa_M(E))} = K.$$

Démonstration.

($\subseteq$) Puisque pour $N \in \mathcal{F}_M$, $\theta^{-1}(N) \in \mathcal{F}_L$ ([58], proposition 4) et que pour $H \in \mathcal{F}_L$, $\kappa_L(H) \in \mathcal{F}_L$ ([58], proposition 2), on a que $K = \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) \in \mathcal{F}_L$ et donc

$$\overline{K \cap \theta^{-1}(\kappa_M(E))} \subseteq \overline{K} = K.$$

($\supseteq$) Soit $s \in K$. Alors, puisque $\mathcal{C}_{hi}(M) \subseteq \mathcal{C}_{lo}(L)$, on a

$$\begin{aligned} \theta_K(s) = \theta(s) \in \theta(K) &= \theta(\kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)})) \\ &= \theta \circ \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) \\ &= \overline{\kappa_M(E)}. \end{aligned}$$

Il existe donc $t \in T^*$ tel que $\theta_K(s)t \in \kappa_M(E)$. Or, pour t ainsi fixée, puisque θ_K est un observateur, on a

$$\begin{aligned} & (\exists\, u \mid u \in \Sigma^* : su \in K \wedge \theta_K(su) = \theta_K(s)t \in \kappa_M(E)) \\ \Longrightarrow\ & (\exists\, u \mid u \in \Sigma^* : su \in K \wedge \theta(su) = \theta_K(su) \in \kappa_M(E)) \\ \Longrightarrow\ & (\exists\, u \mid u \in \Sigma^* : su \in K \wedge su \in \theta^{-1}(\kappa_M(E))) \\ \Longrightarrow\ & (\exists\, u \mid u \in \Sigma^* : su \in K \cap \theta^{-1}(\kappa_M(E))) \\ \Longrightarrow\ & s \in \overline{K \cap \theta^{-1}(\kappa_M(E))} \end{aligned}$$

Sachant que s est arbitraire dans K on a donc $\overline{K \cap \theta^{-1}(\kappa_M(E))} \supseteq K$.
D'où on peut conclure que $\overline{K \cap \theta^{-1}(\kappa_M(E))} = K$.

□

Adaptation de la proposition 16 ([58]).

En supposant que

$\mathcal{C}_{hi}(M) \subseteq \theta(\mathcal{C}_{lo}(L))$;

la restriction de l'équation A.1 est vérifiée ;

et que la paire (L, θ) *est fortement observable par rapport à* $\mathcal{X}$;

alors

$$(\forall\, E \subseteq M)\ \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) = \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}.$$

Démonstration. Prenons $E \subseteq M$.

($\supseteq$) Par consistance hiérarchique restreinte on a

$$\begin{aligned} \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) &= \kappa_L \circ \theta^{-1}(\overline{\theta \circ \kappa_L \circ \theta^{-1}(\kappa_M(E))}) \\ &= \kappa_L \circ \theta^{-1}(\overline{\theta(\kappa_L \circ \theta^{-1}(\kappa_M(E)))}) \\ &= \kappa_L \circ \theta^{-1}(\theta(\overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))})) \end{aligned}$$

puisque θ est causale. Et puisque $\theta^{-1}(\theta(N)) \supseteq N$ en général, on a que

$$\kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) \supseteq \kappa_L(\overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}) = \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}$$

car $\kappa_L \circ \theta^{-1}(\kappa_M(E)) \in \mathcal{C}_{lo}(L)$, ainsi $\overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))} \in \mathcal{C}_{lo}(L)$, et κ_L est idempotente.

($\subseteq$) Posons $K := \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)})$ et $\theta_K := \theta|_K$. On a que

$$\theta(K) = \theta \circ \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) = \overline{\kappa_M(E)} \in \mathcal{C}_{hi}(M)$$

par consistance hiérarchique restreinte, et

$$\kappa_L \circ \theta^{-1}(\theta(K)) = \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) = K$$

et par conséquent $K \in \mathcal{X}$ et θ_K est dotée de la propriété d'observateur. Ainsi d'après le lemme A.1, $\overline{K \cap \theta^{-1}(\kappa_M(E))} = K$. Or puisque $\overline{K \cap \theta^{-1}(\kappa_M(E))} = K \in \mathcal{C}_{lo}(L)$ alors

$$K \cap \theta^{-1}(\kappa_M(E)) \in \mathcal{C}_{lo}(L)$$

en vertu de l'équation A.1.

Maintenant posons $s \in K$ et, par le même argument que celui du lemme, il existe $u \in \Sigma^*$ pour lequel

$$su \in K \cap \theta^{-1}(\kappa_M(E)) \subseteq \theta^{-1}(\kappa_M(E)).$$

Puisque $K \cap \theta^{-1}(\kappa_M(E)) \in \mathcal{C}_{lo}(L)$, on a que $su \in \kappa_L \circ \theta^{-1}(\kappa_M(E))$, et par conséquent $s \in \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}$.

□

Adaptation du théorème 5 ([58]).

En supposant

$\mathcal{C}_{hi}(M) \subseteq \theta(\mathcal{C}_{lo}(L))$;

la restriction de l'équation A.1 ;

$\theta^{-1}(M_{m}) = L_m$;

(L, θ) *fortement observable par rapport à* $\mathcal{X}$;

alors

$$(\forall\, E \in M)\ \kappa_M(E) \textit{ non bloquant} \iff \kappa_L \circ \theta^{-1}(\kappa_M(E)) \textit{ non bloquant.}$$

Démonstration.

Pour $E \subseteq M$ d'après la proposition 16 on a $\kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) = \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))}$, et $\kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) \in \mathcal{X}$. Par conséquent $K := \overline{\kappa_L \circ \theta^{-1}(\kappa_M(E))} \in \mathcal{X}$; ainsi θ_K possède la propriété d'observateur et l'argument du théorème 4 s'applique intégralement ici. □

Adaptation de la proposition 17 ([58]).

Supposons que $\mathcal{C}_{hi}(M) \subseteq \theta(\mathcal{C}_{lo}(L))$ *et que*

θ *possède la propriété d'observateur ;*

$\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$.

Alors

(L, θ) *est fortement observable par rapport à* $\mathcal{X}$.

L'argument de preuve donné dans [58] demeure essentiellement inchangé, une seule étape de preuve invoquant la consistance hiérarchique complète doit être clarifiée.

L'une des étapes de la preuve invoque $\mathcal{C}_{hi}(M) = \theta(\mathcal{C}_{lo}(L))$ pour justifier que $\theta(K) \in \mathcal{C}_{hi}(M)$ pour $K \in \mathcal{X}$. Or dans le contexte défini plus haut, on a déjà

$$\mathcal{X} := \{K \in \overline{\mathcal{C}}_{lo}(L) \mid \kappa_L \circ \theta^{-1} \circ \theta(K) = K,\ \theta(K) \in \mathcal{C}_{hi}(M)\}$$

et ainsi $K \in \mathcal{X} \implies K \in \mathcal{C}_{hi}(M)$. Par conséquent la preuve de la proposition 17 dans [58] s'applique intégralement ici.

Le théorème 6 de Wong et Wonham [58] résume les résultats à partir de la proposition 15, et en tant que tel, ce théorème a déjà été démontré par la démonstration des résultats précédents.

Dans sa forme originale le théorème 6 présume $\mathcal{C}_{hi}(M) = \theta(\mathcal{C}_{lo}(L))$ et se formule comme suit.

Théorème 6 ([58]).

Supposons que la restriction de l'équation A.1 sur $\mathcal{C}_{lo}$ tienne, et que

$\theta^{-1}(M_m) = L_m$;

θ est dotée de la propriété d'observateur ;

$\theta_{|r}^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$.

Alors

$$(\forall E \in M)\ \kappa_M(E) \textit{ non bloquant} \iff \kappa_L \circ \theta^{-1}(E) \textit{ non bloquant.}$$

Dans sa forme adaptée à la consistance hiérarchique restreinte, seule la conclusion du théorème doit être modifiée.

$$(\forall E \in M)\ \kappa_M(E) \textit{ non bloquant} \iff \kappa_L \circ \theta^{-1}(\kappa_M(E)) \textit{ non bloquant}$$

En conclusion, les résultats de Wong Wonham [58] sur la synthèse non bloquante sont aussi applicables à la consistance hiérarchique restreinte.

Annexe B

Quelques propriétés des projections causales

B.1 Propriétés fondamentales

Soit Σ et T deux alphabets (ensembles finis de symboles). Soit $L \subseteq \Sigma^*$ et $M \subseteq T^*$ deux langages préfixes clos. On dit que $\theta : L \to M$ est une *projection causale* si et seulement si pour tout $u, v \in L$ telles que $u \leq v$ alors $\theta(u) \leq \theta(v)$. On peut étendre $\theta : L \to M$ de la façon usuelle à $\theta : \mathcal{P}(L) \to \mathcal{P}(M)$ par $\theta(K) := \{\theta(s) \mid s \in K\}$ pour $K \subseteq L$. Pour θ causale on a alors $\theta(\overline{K}) = \overline{\theta(K)}$ pour tout $K \subseteq L$. Aussi, à $\theta : L \to M$ correspond une fonction de préimage $\theta^{-1} : \mathcal{P}(M) \to \mathcal{P}(L) : N \mapsto K$, où $K = \{s \in L \mid (\exists t \in N)\ \theta(s) = t\}$. Par abus de notation, pour $t \in M$ on utilise souvent $\theta^{-1}(t)$ pour signifier $\theta^{-1}(\{t\}) = \{s \in L \mid \theta(s) = t\}$.

Proposition B.1. *Soit $K_1, K_2 \subseteq L$. Alors $\theta(K_1 \cup K_2) = \theta(K_1) \cup \theta(K_2)$.*

Démonstration. Découle directement de la définition de $\theta : \mathcal{P}(L) \to \mathcal{P}(M)$. □

Proposition B.2. *Soit K_1. $K_2 \subseteq L$. Alors $\theta(K_1 \cap K_2) \subseteq \theta(K_1) \cap \theta(K_2)$.*

Démonstration.

$$\begin{aligned} t \in \theta(K_1 \cap K_2) &\implies (\exists s \mid s \in K_1 \cap K_2 : \theta(s) = t) \\ &\implies (\exists s \mid s \in K_1 : \theta(s) = t) \wedge (\exists s \mid s \in K_2 : \theta(s) = t) \\ &\implies t \in \theta(K_1) \wedge t \in \theta(K_2) \\ &\implies t \in \theta(K_1) \cap \theta(K_2) \end{aligned}$$ □

En général cependant, l'inclusion peut être stricte comme en témoigne l'exemple suivant.

Exemple. *Posons $s_1, s_2 \in L$ ($s_1 \neq s_2$) tels que $\theta(s_1) = \theta(s_2) = t \in M$. Posons aussi $K_1 := \{\epsilon, s_1\}$ et $K_2 := \{\epsilon, s_2\}$. On a donc,*

$$\begin{aligned} \theta(s_1) = t &\implies t \in \theta(K_1) \\ \theta(s_2) = t &\implies t \in \theta(K_2) \end{aligned}$$

et ainsi $t \in \theta(K_1) \cap \theta(K_2)$. Or $t \notin \theta(K_1 \cap K_2) = \theta(\{\epsilon\}) = \{\epsilon\}$.

Le lemme 1 de Wong [55] (page 96) présente un résultat intéressant qui est reproduit ici sous forme d'une proposition dans le but d'en clarifier la preuve.

Proposition B.3. *Soit $H, J \subseteq L$ et supposons que $\theta^{-1}(\theta(H)) = H$ ou que $\theta^{-1}(\theta(J)) = J$. Alors*

$$\theta(H \cap J) = \theta(H) \cap \theta(J).$$

Démonstration. Étant donnée le résultat de la proposition B.2, il suffit de démontrer que $\theta(H \cap J) \supseteq \theta(H) \cap \theta(J)$. Supposons que $\theta^{-1}(\theta(H)) = H$.

$$\begin{aligned} t \in \theta(H) \cap \theta(J) &\implies (\exists s \mid s \in J : \theta(s) = t \wedge s \in \theta^{-1}(\theta(H)) = H) \\ &\qquad \text{car } t \in \theta(H) \text{ et } s \in \theta^{-1}(t) \\ &\implies (\exists s \mid s \in J \cap H : \theta(s) = t) \\ &\implies t \in \theta(J \cap H) \end{aligned}$$ □

Proposition B.4. *Soit* $N_1, N_2 \subseteq M$. *Alors* $\theta^{-1}(N_1 \cup N_2) = \theta^{-1}(N_1) \cup \theta^{-1}(N_2)$ *et donc* θ^{-1} *est monotone par rapport à l'inclusion de langages.*

Démonstration.

$$\begin{aligned} s \in \theta^{-1}(N_1 \cup N_2) &\iff \theta(s) \in N_1 \cup N_2 \\ &\iff \theta(s) \in N_1 \text{ ou } \theta(s) \in N_2 \\ &\iff s \in \theta^{-1}(N_1) \text{ ou } s \in \theta^{-1}(N_2) \\ &\iff s \in \theta^{-1}(N_1) \cup \theta^{-1}(N_2) \end{aligned}$$ □

Proposition B.5. *Soit* $N_1, N_2 \subseteq M$. *Alors* $\theta^{-1}(N_1 \cap N_2) = \theta^{-1}(N_1) \cap \theta^{-1}(N_2)$.

Démonstration.

$$\begin{aligned} s \in \theta^{-1}(N_1 \cap N_2) &\iff \theta(s) \in N_1 \cap N_2 \\ &\iff \theta(s) \in N_1 \text{ et } \theta(s) \in N_2 \\ &\iff s \in \theta^{-1}(N_1) \text{ et } s \in \theta^{-1}(N_2) \\ &\iff s \in \theta^{-1}(N_1) \cap \theta^{-1}(N_2) \end{aligned}$$ □

Proposition B.6. *Soit* $N \subseteq M$. *Alors* $\overline{\theta^{-1}(N)} \subseteq \theta^{-1}(\overline{N})$.

Démonstration. De la proposition B.4, on a que θ^{-1} est monotone par rapport à l'inclusion de langages. Aussi, la fermeture préfixée est idempotente et monotone par rapport à l'inclusion de langages. Enfin $\overline{\theta^{-1}(\overline{N})} = \theta^{-1}(\overline{N})$ puisque $\theta^{-1}(\overline{N}) \in \mathcal{F}_L$ selon la proposition 26 de Wong [55].

$$\begin{aligned} N \subseteq \overline{N} &\implies \theta^{-1}(N) \subseteq \theta^{-1}(\overline{N}) \\ &\implies \overline{\theta^{-1}(N)} \subseteq \overline{\theta^{-1}(\overline{N})} \\ &\implies \overline{\theta^{-1}(N)} \subseteq \theta^{-1}(\overline{N}) \end{aligned}$$ □

B.2 Absence de couplage de contrôle

Dans [58], Wong et Wonham introduisent une propriété qui joue un rôle clef dans la théorie du contrôle hiérarchique non bloquant ; l'*absence de couplage de contrôle* (de l'an-

glais «*generalized partner-freedom condition*»). Une projection causale $\theta : L \twoheadrightarrow M$ est dite libre de couplage de contrôle si et seulement si

$$\theta_v{}^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M \tag{B.1}$$

où $\overline{\kappa}_M := \kappa_{M|\mathcal{F}_M}$.

La notion de couplage de contrôle est présentée par Zhong et Wonham dans [66] ; on y parle de *partenaires* (de l'anglais *partners*) en référence à une situation illustrée dans la figure B.1. Dans le contexte de Zhong et Wonham, l'agent est muni d'une technologie de contrôle standard et θ possède une propriété dite OCC (de l'anglais pour *Output Control Consistency*, à ce sujet voir l'appendice A de [59] pour plus de clarté) de sorte que la technologie de contrôle induite dans l'abstraction est également standard.

Dans la figure B.1, $\theta(s')\tau_1, \theta(s')\tau_2 \in M$ et $\tau_1, \tau_2 \in \Upsilon_c$ de sorte qu'il peut se présenter une situation où $N \in \overline{\mathcal{C}}_{hr}(M)$ et $\theta(s')\tau_1 \in N$ mais $\theta(s')\tau_2 \notin N$ (ou vice-versa). Or la suppression de la génération de ces chaînes à travers l'agent dépend du seul élément de contrôle commun $s \in \Sigma^*$ dans $s's \in L$. Alors pour supprimer la génération de $\theta(s')\tau_2$ (puisque $\theta(s')\tau_2 \notin N$), il faudra supprimer la génération de $s'ss'_2\sigma_2$ en inhibant l'événement contrôlable σ dans le segment $s's$ menant à $s'ss'_2\sigma_2$. Or ceci implique de supprimer aussi la génération de $s'ss'_1\sigma_1$ et par conséquent la génération de la chaîne $\theta(s'ss'_1\sigma_1) = \theta(s')\tau_1$. Dans ce sens, les chaînes $s'ss'_1\sigma_1 \in L$ et $s'ss'_2\sigma_2 \in L$ sont des *partenaires de contrôle* dans l'agent puisque le contrôle de leur génération dépend d'un unique *élément de contrôle commun*, l'événement $\sigma \in \Sigma_c$ du segment $s \in \Sigma^*$ dans la chaîne $s's \in L$.

On peut déjà noter que, dans le contexte de Zhong et Wonham [66], la présence de partenaires de contrôle a pour effet de provoquer une *surestimation* des options de contrôle disponibles dans l'agent par rapport à celles dont dispose l'abstraction. Dans ces conditions, une synthèse dans l'abstraction peut se révéler trop *optimiste* par rapport à ce que l'agent peut accomplir. En clair, il se pourrait que l'agent ne puisse pas générer l'intégralité de l'ensemble des chaînes que contient la synthèse $\overline{\kappa_M(E)}$ mais seulement un sous-ensemble strict de cette dernière, compromettant ainsi la propriété de *consistance hiérarchique* telle que définie dans [66]. La solution est bien sûr de bannir la présence de partenaires de contrôle en établissant la propriété SOCC définie dans [66] (de l'anglais pour *Strict Output*

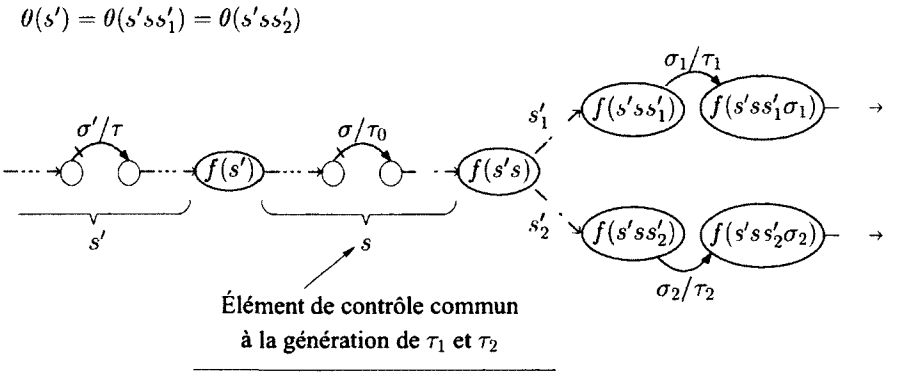


FIGURE B.1 – Couplage de contrôle

Control Consistency).

Une autre façon de formuler ce dilemme est de dire que l'on a un objectif de contrôle abstrait $E \subseteq M$ pour lequel $N = \overline{\kappa_M(E)}$, avec $\theta(s')\tau_1 \in N$ et $\theta(s')\tau_2 \notin N$. Alors $\kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)})$, l'*implémentation sur l'agent du comportement de la synthèse de haut niveau*, ne contiendra ni $s'ss'_1\sigma_1 \in L$ ni $s'ss'_2\sigma_2 \in L$ puisque la génération de $\theta(s')\tau_2$ doit être inhibée. Mais on aura $s'ss'_1\sigma_1 \in \theta_v^{-1}(\overline{\kappa_M(E)})$ puisque $\theta(s')\tau_1 \in N = \overline{\kappa_M(E)}$. En bref on a que $\theta_v^{-1}(\overline{\kappa_M(E)}) \not\subseteq \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)})$ ou encore plus brièvement $\theta_v{}^{-1} \circ \overline{\kappa}_M \not\leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$, et il y a couplage de contrôle.

On note que $\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$ est une propriété plus générale que la simple absence de partenaires au sens de Zhong et Wonham [66]. Dans le contexte de [66], si $\mathcal{P}$ représente un ensemble de couples de chaînes ayant un contrôle commun dans L (des partenaires de contrôle), alors d'après la proposition 106 dans [55]

$$\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M \iff \mathcal{P} = \emptyset$$

Cependant cette caractérisation ne tient que si et seulement si C_{h_i} est munie d'une technologie de contrôle standard (contexte de Zhong et Wonham [66]). Autrement (C_{h_i} est munie d'une technologie de contrôle non standard) on a

$$\mathcal{P} = \emptyset \implies \theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$$

c'est-à-dire que la propriété $\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$ peut quand même être vérifiée en présence de partenaires de contrôle.

Le résultat suivant, tiré de [55] en appendice D, est intéressant en soi mais ne comporte pas de preuve ; nous en avons ajouté une.

Proposition B.7.

$$\begin{aligned} & \theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M \\ \iff \quad & (\forall N \in \overline{\mathcal{C}}_{hi}(M))(\exists K \in \overline{\mathcal{C}}_{lo}(L))\ \theta_v^{-1}(N) \subseteq K \subseteq \theta^{-1}(N). \end{aligned}$$

Démonstration.
($\Longrightarrow$) Si $\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$ alors

$$(\forall E \in \mathcal{F}_M)\ \theta_v^{-1}(\overline{\kappa}_M(E)) \subseteq \kappa_L \circ \theta^{-1}(\overline{\kappa}_M(E)).$$

Puisqu'en général $\kappa_L(\theta^{-1}(\overline{\kappa}_M(E))) \subseteq \theta^{-1}(\overline{\kappa}_M(E))$, on a que

$$(\forall E \in \mathcal{F}_M)\ \theta_v^{-1}(\overline{\kappa}_M(E)) \subseteq \kappa_L \circ \theta^{-1}(\overline{\kappa}_M(E)) \subseteq \theta^{-1}(\overline{\kappa}_M(E)).$$

Or $\overline{\kappa}_M(E) \in \overline{\mathcal{C}}_{hi}(M) \subseteq \mathcal{F}_M$ ce qui implique que $\theta^{-1}(\overline{\kappa}_M(E)) \in \mathcal{F}_L$ (proposition 26 dans [55]). Et puisque κ_L préserve la fermeture préfixée on a que $\kappa_L \circ \theta^{-1}(\overline{\kappa}_M(E)) \in \overline{\mathcal{C}}_{lo}(L)$.
($\Longleftarrow$) Supposons

$$(\forall N \in \overline{\mathcal{C}}_{hi}(M))(\exists K \in \overline{\mathcal{C}}_{lo}(L))\ \theta_v^{-1}(N) \subseteq K \subseteq \theta^{-1}(N).$$

Puisque pour tout $E \in \mathcal{F}_M$, $\kappa_M(E) = \overline{\kappa}_M(E) \in \overline{\mathcal{C}}_{hi}(M)$, alors

$$(\forall E \in \mathcal{F}_M)(\exists K \in \overline{\mathcal{C}}_{lo}(L))\ \theta_v^{-1}(\overline{\kappa}_M(E)) \subseteq K \subseteq \theta^{-1}(\overline{\kappa}_M(E)).$$

Or puisque $K \subseteq \theta^{-1}(\overline{\kappa}_M(E))$ et que κ_L est monotone par rapport à l'inclusion d'ensembles, on a que $K = \kappa_L(K) \subseteq \kappa_L(\theta^{-1}(\overline{\kappa}_M(E)))$ et en conséquence

$$(\forall E \in \mathcal{F}_M)\ \theta_v^{-1}(\overline{\kappa}_M(E)) \subseteq \kappa_L(\theta^{-1}(\overline{\kappa}_M(E))).$$ □

B.3 Coïncidence du contrôle

Posons $L \subseteq \Sigma^*$ et $M \subseteq T^*$ fermés, $\mathcal{C}_{lo}$ et $\mathcal{C}_{hi}$ deux structures de contrôle sur L et M respectivement et $\theta : L \rightarrow M$ causale. Supposons que, sur le triplet (L, M, θ), on a la propriété suivante :

$$\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L) \tag{B.2}$$

c'est-à-dire que la préimage de tout sous-langage contrôlable de l'*abstraction* est aussi contrôlable dans l'*agent*. Cette propriété est une généralisation de la notion d'*absence de délai de contrôle* originalement formulée par Zhong et Wonham dans [65]. Dans le contexte original de Zhong et Wonham, l'*absence de délai de contrôle* présuppose que $\mathcal{C}_{lo}$ et $\mathcal{C}_{hi}$ sont toutes deux munies d'une technologie de contrôle standard ($\Sigma = \Sigma_c \dot{\cup} \Sigma_u$ et $T = T_c \dot{\cup} T_u$). Dans [55] et [59] Wong et Wonham généralisent le concept à des structures de contrôle arbitraires sans toutefois lui donner de nom spécifique. Pour distinguer entre les deux concepts on peut désigner la propriété définie par l'équation B.2 en tant que *coïncidence du contrôle*.

Tel que mentionné précédemment la *coïncidence du contrôle* prend son origine dans le concept d'*absence de délai de contrôle*. Or comme mentionné dans [65] l'absence de délai de contrôle est nécessaire pour assurer que la *composition synchrone* de deux modèles *abstraits* (au sens du contrôle hiérarchique de la section 2.3.2) ne détruise pas les propriétés de la projection θ (entre autres la consistance du contrôle introduite dans la section 2.3.2.2). Pour l'élaboration de composantes réutilisables, dont les *modèles génériques* sont des *abstractions* conformes au modèle du contrôle hiérarchique, il importe de considérer ces deux propriétés et de comprendre les problèmes liés à la présence de délai de contrôle dans ces modèles. La figure B.2 donne une illustration de ce que représente un *délai de contrôle*. Cet exemple suppose le contexte présenté dans [65]. En particulier les deux niveaux (abstraction et agent) sont dotés d'une technologie de contrôle standard. Dans l'illustration on voit que le contrôle de la génération de l'événement contrôlable $\tau \in T_c$ par l'abstraction dépend d'un élément de contrôle $\sigma_p \in \Sigma_c$ dans l'agent.

Dans la figure B.2 il est évident qu'une fois franchie la transition (n_p, σ_p, n_{p-1}) dans l'agent, la génération de $\tau \in T_c$ dans l'abstraction est inévitable puisque $s' \in \Sigma_u^+$. Or l'abstraction ne sera *informée* du fait que l'agent a franchi la transition (n_p, σ_p, n_{p-1}) que lors de la génération de l'événement τ par la projection, c'est-à-dire lors de la transition (n_1, σ_1, n).

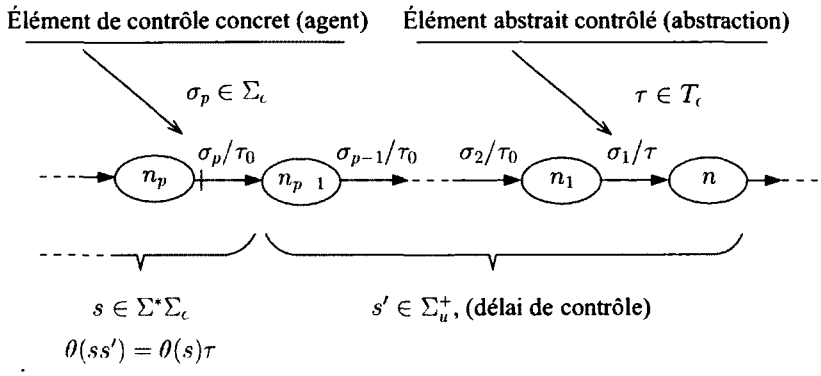


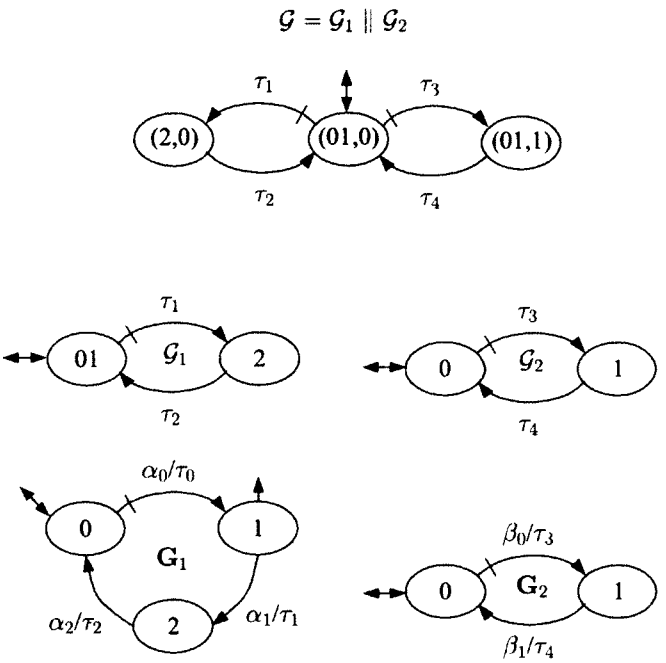
FIGURE B.2 – Délai de contrôle

Il y a donc un *délai* relatif à l'*information* disponible dans l'abstraction, par rapport à celle qui est disponible dans l'agent, en ce qui concerne le contrôle de la transition (n_p, σ_p, n_{p-1}). En l'absence de parallélisme effectif entre ce système et un autre système, c'est-à-dire si le système considéré est pris en *isolation*, cette situation n'est pas catastrophique. Comme mentionné dans [65] le système peut être défini comme dans la figure B.2 et toujours posséder la propriété SOCC (*Strict Output Control Consistency*, voir aussi [62]) qui en garantit la *consistance hiérarchique* (et donc la consistance du contrôle au sens du théorème 1 dans [58]). Cependant aussitôt que l'on veut mettre ce système en relation d'*exécution parallèle concurrente* (c'est-à-dire utiliser une composition par produit synchrone telle que définie dans la section 1.1.4) la situation change.

La figure B.3 illustre deux systèmes simples complètement disjoints, c'est-à-dire $(\mathbf{G}_i, \mathcal{G}_i)$ $(i = 1, 2)$ qui ne partagent aucun de leurs événements, ni dans leurs agents $(\mathbf{G}_i)$ ni dans leurs abstractions $(\mathcal{G}_i)$ respectives et en plus $\Sigma_i \cap T_i = \emptyset$ pour $i = 1, 2$. On peut aisément vérifier que $\mathcal{G}_i$ est un observateur naturel de $\mathbf{G}_i$ pour $i = 1, 2$, et aussi que chacun de ces systèmes est SOCC dans les termes de Zhong et Wonham [65]. On note qu'il y a un *délai de contrôle* dans la paire $(\mathbf{G}_1, \mathcal{G}_1)$ relativement à la génération de l'événement $\tau_1 \in T_1$ dans $\mathcal{G}_1$ qui est contrôlée par la génération de $\alpha_0 \in \Sigma_1$ dans l'agent $\mathbf{G}_1$.

L'effet du *délai de contrôle* devient apparent, dans la figure B.3, lorsque l'on considère la composition synchrone des abstractions, $\mathcal{G} = \mathcal{G}_1 \parallel \mathcal{G}_2$, et qu'on la contraste avec la composition synchrone des agents $\mathbf{G} = \mathbf{G}_1 \parallel \mathbf{G}_2$.

Dans les termes de [65], on perd non seulement la propriété SOCC (et donc la consis-



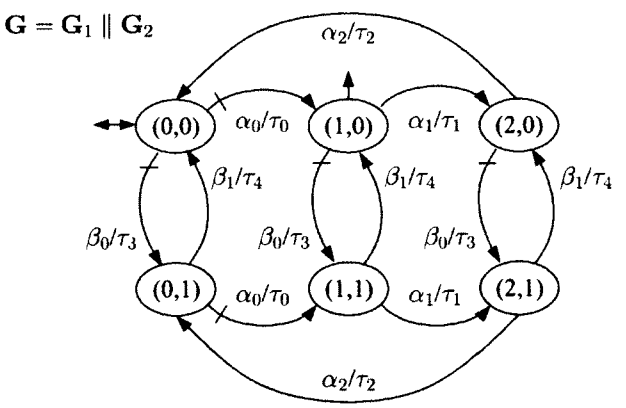


FIGURE B.3 – Effet d'un délai de contrôle

tance hiérarchique) mais la projection n'est même plus OCC (*Output Control Consistent*) et n'a donc plus de technologie de contrôle consistante. Par exemple, dans $\mathcal{G}$, $\tau_4 \in T_u$; or dans G on peut générer τ_4 (à travers θ) d'au moins deux façons :

- $\theta(\beta_0\beta_1) = \tau_3\tau_4$, dans ce cas la génération de τ_4 est effectivement incontrôlable ;
- $\theta(\beta_0\alpha_0\beta_1) = \tau_3\tau_4$ et dans ce cas, la génération de τ_4 peut être contrôlée à travers $\alpha_0 \in \Sigma_c$.

Alors est-ce que τ_4 est contrôlable ou non ? Dans le contexte de [65], en présence de délai de contrôle, le produit synchrone peut simplement détruire les conditions requises pour la consistance hiérarchique. La situation est similaire pour le contexte défini par Wong et Wonham [59] (en appendice A) puisque la propriété OCC entre G et $\mathcal{G}$ ne peut être assurée.

De ce qui précède on peut voir que la coïncidence du contrôle est une propriété importante si on veut utiliser la composition synchrone d'abstractions. On trouve en annexe C une caractérisation de la propriété d'absence de délai de contrôle (formule C.6). Aussi, dans la même section, la proposition C.11 montre que l'absence de délai de contrôle entraîne la coïncidence du contrôle, et la proposition C.14 montre que l'absence de délai de contrôle est préservée par le produit synchrone.

B.4 Lois de contrôle synchrones

D'après Wong [55], pour que l'on puisse calculer une loi de contrôle sur un agent à partir d'une loi de contrôle sur l'abstraction qui lui correspond, ces lois de contrôle doivent être *synchrones*. Il faut d'abord se pencher sur cette notion.

Pour une loi de contrôle $V : L \rightarrow \mathcal{P}(\Sigma)$, $V(s) \in \mathcal{P}(\Sigma)$ donne l'ensemble des événements dont la génération est *permise par contrôle* après la génération de $s \in L$. Sa complémentaire $V^c(s) := \Sigma - V(s)$ donne l'ensemble des événements à inhiber à chaque point de l'évolution d'un système. Supposons donc un système correspondant au modèle du contrôle hiérarchique de la section 2.3.2 et considérons $V_K^c(s)$ et $V_N^c(t)$, pour $s \in L$ et $t \in M$, les complémentaires de V_K et V_N respectivement. Alors, au moment où l'agent génère $s \in \theta_v^{-1}(t)$, l'abstraction se retrouve en $t \in \overline{N}$ et l'ensemble des événements de $V_N^c(t)$ sont inhibés dans l'abstraction. Tant que l'agent se trouve dans $\theta^{-1}(t)$ la configuration de contrôle de l'abstraction ne change pas, tandis que $V_K^c(s)$ varie selon $s \in \theta^{-1}(t)$.

Informellement, on dit que deux lois de contrôle V_K et V_N (sur l'agent et l'abstraction respectivement) sont *synchrones* si l'*effet* de l'ensemble des $V_K^c(s)$ pour $s \in \theta^{-1}(t)$ est précisément d'inhiber la génération, dans l'abstraction, de toutes chaînes $t\tau \in M$ telles que $\tau \in V_N^c(t)$.

D'après la proposition 86 dans [55], en présence de consistance hiérarchique, pour $E \subseteq M$, $N := \kappa_M(E)$ non vide et $K := \kappa_L \circ \theta^{-1}(E)$, V_K et V_N existent et on a alors que si

θ est dotée de la propriété d'observateur ;

$\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$;

alors V_K et V_N sont synchrones.

Rappelons brièvement que pour un langage M défini sur un alphabet T,

$$Elig_M : M \to \mathcal{P}(T) : t \mapsto \{\tau \in T \mid t\tau \in M\}.$$

$Elig_M(t)$ donne donc l'ensemble des événements que le système peut générer immédiatement après la génération d'une chaîne $t \in M$.

À l'instar de Wong [55] en section 6.3, on définit

$$f_v : L_{voc} \to \mathcal{P}(L_{voc}) : s \mapsto pos_L(s) \cap \omega^{-1}(Elig_M(\theta(s))).$$

Pour se situer dans l'argumentation précédente avec $s \in \theta_v^{-1}(t)$, $f_v(s)$ est formé de toutes les chaînes $sv \in L_{voc} \subseteq L$ telles que $\theta(sv) = \theta(s)\tau = t\tau \in M$. Il s'agit donc de toutes les extensions de $s \in L$ (dans l'agent) qui peuvent provoquer la génération d'un événement $\tau \in T$ à la suite de la génération de $t \in M$ dans l'abstraction. On peut remarquer que $f_v(s) \subseteq L_{voc} \subseteq L$ est un sous-langage de L et, puisque L est préfixe clos par définition, $H_s := \overline{f_v(s)}$ est aussi un sous-langage fermé de L. Or, puisque C_{lo} est une structure de contrôle sur L, on a que $C_{lo}(H_s)$ est bien définie et par conséquent on dispose d'un opérateur de synthèse $\kappa_{H_s} : \mathcal{P}(H_s) \to C_{lo}(H_s)$. En effet H_s constitue un sous-système de L en ce sens que dans l'arbre d'accessibilité $\widehat{\mathbf{T}}$ de la section 2.3.3.4, H_s est une sous-structure de transition et donc un *sous-système* de $\widehat{\mathbf{T}}$ (ceci est le sens donné au terme *sous-système* dans [55]). On peut considérer que H_s forme une *cellule* de $\widehat{\mathbf{T}}$. Supposons pour l'instant que $H_s \neq \emptyset$.

Définissons maintenant

$$g_v : L_{voc} \to \mathcal{P}(L_{voc}) : s \mapsto pos_L(s) \cap \omega^{-1}(V_N^c(\theta(s))).$$

En des termes similaires aux précédents, $g_v(s)$ est formé de toutes les chaînes $sv \in L_{voc}$ telles que $\theta(sv) = \theta(s)\tau = t\tau \in M$ et $\tau \in V_N^c(t)$. C'est-à-dire qu'il s'agit de toutes les extensions de s dans L dont la génération doit être empêchée dans l'agent pour que la projection demeure conforme à $L(V_N)$ dans l'abstraction.

On peut ainsi former un *objectif de contrôle* $E_s := H_s - g_v(s)$ local à la cellule H_s. Alors $K_s := \kappa_{H_s}(E_s)$ est une solution partielle (locale à H_s) au problème de déterminer V_K car $K_s \subseteq E_s$ est contrôlable par rapport à L dans H_s. Il convient de noter que par construction $g_v(s) \subseteq f_v(s) \subseteq L_{voc} \subseteq L$. Aussi, si $sv \in g_v(s)$ on a $s \in L(V_K)$ et $\theta(s) = t \in M(V_N)$, alors le contrôle à appliquer s'exerce nécessairement sur un préfixe $u \leq v$ de la sous-chaîne v à l'intérieur de la cellule H_s. Autrement dit, si on déterminait V_{K_s} comme loi de contrôle partielle dans H_s, il est impossible que cette dernière interfère avec le contrôle d'une cellule précédent H_s ou au-delà de H_s, c'est-à-dire *à l'extérieur* de la cellule H_s. On peut donc, du moins conceptuellement, calculer V_K *cellule* par *cellule* (V_{K_s} pour $s \in \overline{K}$) à partir de $t \in M(V_N)$, V_N, f_v et g_v. Les résultats de la section 6.3 de [55] développent formellement cette idée en prouvant que dans les conditions du théorème 6 ([55]), V_K égale la somme des *morceaux* V_{K_s}, notamment

$$\bigcup_{s \in L_{voc} \cap \overline{K}} \kappa_{H_s}(E_s) = \overline{\kappa_L \circ \theta^{-1}(E)} = \kappa_L \circ \theta^{-1}(\overline{\kappa_M(E)}) = \overline{K}.$$

Cette solution conceptuelle n'est cependant praticable que pour des générateurs finis bien sûr.

Pour illustrer le calcul prenons le contexte de Zhong et Wonham [66] où l'agent ainsi que l'abstraction sont tous deux munis de technologies de contrôle standards et la projection est SOCC (c'est-à-dire OCC sans couplage du contrôle). On a donc $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$ pour l'agent et $T = T_c \dot{\cup} T_u$ pour l'abstraction. Comme mentionné précédemment, si $sv \in g_v(s)$, puisque $g_v(s) \subseteq f_v(s)$, un contrôle devra s'appliquer quelque part sur $u \leq v$ un préfixe de v, et donc sur un certain $\sigma \in \Sigma_c$ pour $v = u'\sigma u''$. Or l'opérateur de synthèse κ_{H_s} a comme

caractéristique de sélectionner le plus grand sous-langage contrôlable. Ceci implique que même s'il existe plusieurs solutions, la solution retenue aura la forme $v = u'\sigma u''$ où $u'' \in \Sigma_u^*$, c'est-à-dire qu'elle sélectionnera le plus grand préfixe contrôlable de v.

Ce qui précède nous donne une caractérisation relativement simple de V_K (section 6.4 dans [55] et section 5.4 dans [62])

$$V_K(w) = \Sigma_u \cup \{\sigma \in \Sigma_c \mid (\forall w' \in \Sigma_u^*)(\forall \tau \in T)\ \theta(w\sigma w') = \theta(w)\tau \implies \tau \in V_N(\theta(w))\}$$

pour $w \in \overline{K}$. Par rapport à ce qui précède $w = su'$ et $w' = u''$. Pour paraphraser ce résultat on peut dire que V_K inhibe toujours le dernier événement contrôlable de toute séquence (d'un sous-système de l'agent) menant à la génération d'un événement inhibé par la loi de contrôle de l'abstraction.

B.5 Projections et systèmes disjoints

Lemme B.8. *Soit $\Sigma := \Sigma_1 \cup \Sigma_2$, $\Sigma_1 \cap \Sigma_2 = \emptyset$ et $p_i : \Sigma^* \to \Sigma_i^*$ ($i = 1, 2$) les projections naturelles correspondantes. Soit $K_i \in \Sigma_i^*$ ($i = 1, 2$) deux langages arbitraires. Alors pour $K := K_1 \parallel K_2$ on a*

$$p_i(K) = K_i\ (i = 1, 2).$$

Démonstration.

($\subseteq$) Puisqu'en général, pour $i \in \{1, 2\}$, $p_i(A \cap B) \subseteq p_i(A) \cap p_i(B)$ (voir entre autres [14] sur les propriétés des projections naturelles) on a

$$\begin{aligned} p_i(K) = p_i(p_1^{-1}(K_1) \cap p_2^{-1}(K_2)) &\subseteq p_i(p_1^{-1}(K_1)) \cap p_i(p_2^{-1}(K_2)) \\ &\subseteq p_i(p_i^{-1}(K_i)) = K_i \end{aligned}$$

($i = 1, 2$).

($\supseteq$) Soit $i, j \in \{1, 2\}$ tels que $i \neq j$ fixés et $s_i \in K_i$, alors $p_j(p_i^{-1}(s_i)) = \Sigma_j^* \supseteq K_j$ puisque $\Sigma_1 \cap \Sigma_2 = \emptyset$. Donc $(\exists\ s \in p_i^{-1}(s_i))\ :\ p_j(s) \in K_j$ et alors $s \in p_j^{-1}(K_j)$ et puisque $s \in p_i^{-1}(s_i) \subseteq p_i^{-1}(K_i)$ on a $s \in p_i^{-1}(K_i) \cap p_j^{-1}(K_j) = K$ et donc

$s_i \in p_i(K)$. Puisque s_i est arbitraire dans K_i on a que $K_i \subseteq p_i(K)$.
On peut donc conclure que $p_i(K) = K_i$ $(i = 1, 2)$. □

Lemme B.9. *Soit* $\Sigma := \Sigma_1 \cup \Sigma_2$, $\Sigma_1 \cap \Sigma_2 = \emptyset$ *et* $p_i : \Sigma^* \to \Sigma_i^*$ $(i = 1, 2)$ *les projections naturelles correspondantes. Soit* $K_i \in \Sigma_i^*$ $(i = 1, 2)$ *deux langages arbitraires.*

Alors pour $K := K_1 \parallel K_2$ *on a* $\overline{K} = \overline{K_1} \parallel \overline{K_2}$.

C'est-à-dire que la synchronisation *de* K_1 *et* K_2 *est* modulaire. *On dit aussi que la* synchronisation *de* K_1 *et* K_2 *est* non conflictuelle *(de l'anglais* synchronously non-conflictual *voir [62]).*

Démonstration. Il faut vérifier

$$\overline{K} = \overline{p_1^{-1}(K_1) \cap p_2^{-1}(K_2)} = \overline{p_1^{-1}(K_1)} \cap \overline{p_2^{-1}(K_2)} = \overline{K_1} \parallel \overline{K_2}$$

donc

$$p_1^{-1}(\overline{K_1}) \cap p_2^{-1}(\overline{K_2}) = \overline{p_1^{-1}(K_1)} \cap \overline{p_2^{-1}(K_2)} \subseteq \overline{p_1^{-1}(K_1) \cap p_2^{-1}(K_2)}$$

puisqu'en général

$$\overline{p_1^{-1}(K_1) \cap p_2^{-1}(K_2)} \subseteq \overline{p_1^{-1}(K_1)} \cap \overline{p_2^{-1}(K_2)}$$

et, pour $i \in \{1, 2\}$, $\overline{p_i{}^{-1}(K_i)} = p_i{}^{-1}(\overline{K_i})$ (voir [14] sur les propriétés des projections naturelles). Autrement dit il faut vérifier que $p_i{}^{-1}(K_i)$ $(i = 1, 2)$ sont *modulaires* (on dit aussi *non conflictuels* voir la section 2.1).

Soit $s \in p_1^{-1}(\overline{K_1}) \cap p_2^{-1}(\overline{K_2})$, alors

$$\begin{aligned} s \in p_i^{-1}(\overline{K_i}) &\implies p_i(s) \in \overline{K_i} \\ &\implies (\exists\, v_i \in \Sigma_i^*) : p_i(s)v_i = p_i(sv_i) \in K_i \end{aligned}$$

$(i = 1, 2)$.

Soit donc $i, j \in \{1, 2\}$ tels que $i \neq j$ fixés, et $p_i(sv_i) \in K_i$, $p_j(sv_j) \in K_j$ définies comme précédemment. Alors $p_j(p_i^{-1}(p_i(sv_i))) = \Sigma_j^* \supseteq K_j$, puisque $\Sigma_1 \cap \Sigma_2 = \emptyset$, et donc $p_i^{-1}(p_i(sv_i)) \cap p_j^{-1}(p_j(sv_j)) \neq \emptyset$. Ainsi il existe $sv \in p_i^{-1}(p_i(sv_i)) \cap p_j^{-1}(p_j(sv_j))$ pour lequel $p_i(sv) = p_i(s)v_i = p_i(sv_i) \in K_i$ et $p_j(sv) = p_j(s)v_j = p_j(sv_j) \in K_j$. Mais alors $sv \in p_i{}^{-1}(K_i) \cap p_j{}^{-1}(K_j)$ et donc $s \in \overline{p_i{}^{-1}(K_i) \cap p_j{}^{-1}(K_j)}$.

Puisque s est arbitraire dans $p_1^{-1}(\overline{K_1}) \cap p_2^{-1}(\overline{K_2})$, il en découle que

$$p_1^{-1}(\overline{K_1}) \cap p_2^{-1}(\overline{K_2}) \subseteq \overline{p_1^{-1}(K_1) \cap p_2^{-1}(K_2)}$$

c'est-à-dire $p_i^{-1}(K_i)$ $(i = 1, 2)$ sont *modulaires* et plus généralement

$$\overline{K} = \overline{p_1^{-1}(K_1) \cap p_2^{-1}(K_2)} = \overline{p_1^{-1}(K_1)} \cap \overline{p_2^{-1}(K_2)} = \overline{K_1} \parallel \overline{K_2}$$

□

Soit $\Sigma := \Sigma_1 \cup \Sigma_2$, $\Sigma_1 \cap \Sigma_2 = \emptyset$ et $q_i : \Sigma^* \to \Sigma_i^*$ $(i = 1, 2)$ les projections naturelles correspondantes (voir la section 1.1.2). Soit $T := T_1 \cup T_2$, $T_1 \cap T_2 = \emptyset$ et $P_i : T^* \to T_i^*$ $(i = 1, 2)$ les projections naturelles correspondantes. On présume aussi que $\Sigma_i \cap T_i = \emptyset$ $(i = 1, 2)$. Soit $L := L_1 \parallel L_2$ avec $L_i \in \Sigma_i^*$ $(i = 1, 2)$ des langages préfixe clos.

Soit $\theta_i : L_i \to T_i^*$ $(i = 1, 2)$ des projections causales définies comme suit

$$\begin{aligned} \theta_i(\varepsilon) &= \varepsilon \\ \theta_i(s\sigma) &= \begin{cases} \theta_i(s) \cdot \omega_i(s\sigma). & \text{si } \omega_i(s\sigma)! \\ \theta_i(s), & \text{autrement} \end{cases} \end{aligned}$$

pour $s\sigma \in L_i$ et $\omega_i : L_i \to T_i$ $(i = 1, 2)$ des fonctions partielles arbitraires. Soit $\theta : L \to T^*$ définie de la façon suivante

$$\begin{aligned} \theta(\varepsilon) &= \varepsilon \\ \theta(s\sigma) &= \begin{cases} \theta(s) \cdot \omega(s\sigma), & \text{si } \omega(s\sigma)! \\ \theta(s), & \text{autrement} \end{cases} \end{aligned}$$

avec $\omega : L \to T$ définie comme suit

$$\omega(s\sigma) := \begin{cases} \omega_i(q_i(s)\sigma), & \text{si } \sigma \in \Sigma_i \text{ et } \omega_i(q_i(s)\sigma)! \ (i = 1, 2) \\ \textit{indéfinie}, & \text{autrement} \end{cases}$$

pour $s\sigma \in L$. D'après Pu [42], dans les conditions précitées, cette définition de θ constitue

la synchronisation de θ_1 et θ_2 notée $\theta := \theta_1 \parallel \theta_2$ et on a alors :

$$\begin{aligned} P_i \circ \theta &= \theta_i \circ q_i \quad i = 1, 2 \\ q_i \circ \theta^{-1} &= \theta_i^{-1} \circ P_i \quad i = 1, 2 \\ \theta(L) &= \theta_1(L_1) \parallel \theta_2(L_2) \end{aligned}$$

correspondant respectivement au lemme A.7, au lemme A.8 et à la proposition A.5 dans [42].

Lemme B.10. *Dans les conditions définies précédemment, $\theta := \theta_1 \parallel \theta_2$ est causale c'est-à-dire*

$$(\forall u, s \in L) \cdot u \leq s \implies \theta(u) \leq \theta(s)$$

Démonstration. En posant $s = uv \in L$ ce qui suit est valide pour $i \in \{1, 2\}$.

$$\begin{aligned} P_i(\theta(s)) &= \theta_i(q_i(s)) \\ &= \theta_i(q_i(uv)) \\ &= \theta_i(q_i(u) \cdot q_i(v)) \\ &= \theta_i(q_i(u)) \quad [\theta_i(q_i(s))/\theta_i(q_i(u))] \\ &= P_i(\theta(u)) \cdot [P_i(\theta(s))/P_i(\theta(u))] \\ &= P_i(\theta(u)) \cdot P_i(\theta(s)/\theta(u)) \\ &= P_i(\theta(u) \cdot \theta(s)/\theta(u)) \,. \end{aligned}$$

C'est-à-dire, sachant que q_i est caténative et que θ_i est causale, on a sûrement que

$$\theta_i(q_i(u)) \leq \theta_i(q_i(uv)) = \theta_i(q_i(s)) \,.$$

Donc $\theta_i(q_i(s)) = \theta_i(q_i(u))t_i$ où $t_i = \theta_i(q_i(s))/\theta_i(q_i(u)) \in T_i^*$, c'est-à-dire que t_i est le suffixe de T_i^* qui doit être ajouté à $\theta_i(q_i(u))$ pour former $\theta_i(q_i(s))$.

Aussi, $\theta(u).\theta(s) \in \theta(L)$ car $s = uv \in L$. On peut donc réécrire t_i comme suit

$$t_i = \theta_i(q_i(s))/\theta_i(q_i(u)) = P_i(\theta(s))/P_i(\theta(u)) = P_i(\theta(s)/\theta(u))$$

en utilisant le lemme A.7 dans [42]. Alors clairement

$$P_i(\theta(s)) = P_i(\theta(u) \cdot \theta(s)/\theta(u))$$

d'où $\theta(u) \leq \theta(s)$ et puisque s est arbitraire dans L, $\theta := \theta_1 \parallel \theta_2$ est causale. □

Lemme B.11. *Dans les conditions définies précédemment.*

$$\theta(L_m) = \theta_1(L_{1,m}) \parallel \theta_2(L_{2,m})$$

pour $L_{i,m} \subseteq L_i$ *tels que* $\overline{L_{i,m}} = L_i$ *(*$i = 1, 2$*) et* $L_m := L_{1,m} \parallel L_{1,m}$.

Démonstration. Puisque θ, θ_1 et θ_2 sont causales

$$\theta(\overline{L_m}) = \theta_1(\overline{L_{1,m}}) \parallel \theta_2(\overline{L_{2,m}}) = \overline{\theta_1(L_{1,m})} \parallel \overline{\theta_2(L_{2,m})} = \overline{\theta(L_m)} .$$

Puisque $T_1 \cap T_2 = \emptyset$

$$\overline{\theta_1(L_{1,m})} \parallel \overline{\theta_2(L_{2,m})} = \overline{\theta_1(L_{1\,m}) \parallel \theta_2(L_{2,m})} = \overline{\theta(L_m)}$$

d'où

$$\theta(L_m) = \theta_1(L_{1,m}) \parallel \theta_2(L_{2,m}) .$$

□

Plus généralement on a que pour $K_i \subseteq L_i$ $(i = 1, 2)$ et $K := K_1 \parallel K_2$

$$\theta(K) = \theta_1(K_1) \parallel \theta_2(K_2) .$$

La consistance du marquage est préservée par le produit synchrone de projections causales de systèmes disjoints.

Lemme B.12. *Dans les conditions définies précédemment, pour* $L_{i,m} \subseteq L_i$ *tels que* $\overline{L_{i,m}} = L_i$ *(*$i = 1, 2$*) et* $L_m := L_{1,m} \parallel L_{2,m}$, *si* $\theta_i^{-1}(\theta_i(L_{i,m})) = L_{i,m}$ *(*$i = 1, 2$*) alors*

$$\theta^{-1}(\theta(L_m)) = L_m .$$

Démonstration. Puisque $\overline{L_{i,m}} = L_i$ $(i = 1, 2)$

$$\theta(\overline{L_m}) = \theta_1(\overline{L_{1,m}}) \parallel \theta_2(\overline{L_{2,m}}) = \theta_1(L_1) \parallel \theta_2(L_2) = \theta(L)$$

le lemme A.8 dans [42] est applicable, ainsi

$$q_i \circ \theta^{-1} = \theta_i^{-1} \circ P_i$$

$(i = 1, 2)$.

Soit donc $t \in \theta(L_m) = \theta_1(L_{1,m}) \parallel \theta_2(L_{2,m})$, alors $t \in P_i^{-1}(\theta_i(L_{i,m}))$ $(i = 1, 2)$. Ainsi $P_i(t) \in \theta_i(L_{i,m})$ $(i = 1, 2)$ et $\theta_i^{-1}(P_i(t)) \subseteq \theta_i^{-1}(\theta_i(L_{i,m})) = L_{i,m}$ $(i = 1, 2)$ puisque $\theta_i^{-1}(\theta_i(L_{i,m})) = L_{i,m}$ $(i = 1, 2)$. Alors, par le lemme A.8 dans [42], $q_i(\theta^{-1}(t)) \subseteq L_{i,m}$ d'où $\theta^{-1}(t) \subseteq q_i^{-1}(L_{i,m})$ $(i = 1, 2)$. Donc

$$\theta^{-1}(t) \subseteq q_1^{-1}(L_{1,m}) \cap q_2^{-1}(L_{2,m}) = L_{1,m} \parallel L_{2,m} = L_m$$

et ainsi $\theta^{-1}(t) \subseteq L_m$.

Puisque t est arbitraire dans $\theta(L_m)$ on a que $\theta^{-1}(\theta(L_m)) \subseteq L_m$. Enfin, puisque $L_m \subseteq \theta^{-1}(\theta(L_m))$ en général, on a $\theta^{-1}(\theta(L_m)) = L_m$. □

B.6 Composition fonctionnelle d'observateurs

Le résultat suivant, dont on trouve une démonstration dans un article dû à Wong et al. [57], nous assure aussi que la composition fonctionnelle de projections causales dotées de la propriété d'observateur préserve la propriété d'observateur. La propriété d'observateur peut être propagée *verticalement*, procurant une forme de *réabstraction* d'une combinaison de modèles eux-mêmes déjà abstraits.

Proposition B.13 (Lemme 5 dans [57]). *Soit Σ, T et Γ des alphabets et $L \subseteq \Sigma^*$ fermé. Soit $\theta : L \to T^*$ et $\varphi : M \to \Gamma^*$ pour $M := \theta(L)$ deux projections causales. Posons $H \subseteq L$ et supposons que θ soit un H-observateur et φ un $\theta(H)$-observateur. Alors $\psi := \varphi \circ \theta$ (donc $\psi : L \to \Gamma^*$) est un H-observateur.*

Annexe C

Abstraction en contrôle hiérarchique

Dans le cadre général de notre problème de conception de composantes réutilisables, le point de départ est toujours un générateur G avec $L = L(\mathrm{G})$, où $L \subseteq \Sigma^*$ est un langage préfixe clos muni d'une technologie de contrôle standard $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$. Cette technologie de contrôle induit $\mathcal{C}_{lo} : \mathcal{F}_L \rightarrow \mathcal{P}^2(L)$ une structure de contrôle standard localement définissable sur L (Wong [55], section 3.1).

Dans ce contexte il est réaliste de supposer que l'on puisse concevoir $\theta : L \rightarrow M$ causale (section 2.3.2.1) pour $M \subseteq T^*$ fermé, $\Sigma \cap T = \emptyset$, telle que M soit aussi muni d'une technologie de contrôle standard $T = T_c \dot{\cup} T_u$ (section 2.3.1.3). Cette dernière induit $\mathcal{C}_{hi}(M)$, aussi une structure de contrôle standard localement définissable sur M (voir la section 2.3.1.2). Il est aussi raisonnable d'envisager que θ soit conçue de sorte à assurer la coïncidence du contrôle (section B.3), $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$, assurant du même coup une forme de consistance du contrôle (section 2.3.2.2) ainsi que l'absence de couplage de contrôle $\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$ (section B.2). Finalement, pour la synthèse non bloquante, il est généralement nécessaire que θ soit aussi dotée de la propriété d'observateur (L_m-observateur, section 2.3.3.5).

C.1 Consistance du contrôle

Dans les conditions précitées, l'intuition suggère qu'il existe $\mathcal{C}'_{lo} : \mathcal{F}_L \rightarrow \mathcal{P}^2(L)$, une structure de contrôle telle que $\mathcal{C}_{lo} \leq \mathcal{C}'_{lo}$ et pour laquelle $\theta^{-1}(\mathcal{C}_{hi}(M)) = \mathcal{C}'_{lo}(L)$. C'est le

cas par exemple d'une projection naturelle (section 1.1.2) où $\Sigma_c \cap \Sigma_o \neq \emptyset$ et $\Sigma_c \not\subseteq \Sigma_o$. Une telle projection ne peut sélectionner qu'une partie de l'ensemble des sous-langages contrôlables de L ($\mathcal{C}_{lo}(L)$) car elle *efface* certains des événements contrôlables de Σ, les soustrayant ainsi à l'action du contrôle à travers la projection. Cependant le langage projeté demeure susceptible à l'action d'un contrôle par inhibition à travers l'ensemble des événements contrôlables effectivement projetés. Aussi lorsque la projection est *calculée* de manière à être dotée de la propriété d'observateur, la synthèse non bloquante sur la projection est alors possible (voir la notion d'*observateurs naturels* dans [14, 15]). Une situation similaire se présente dans le cas des *projections trouées* (de l'anglais *forgetful maps*) étudiées dans [57]. Bien sûr dans ces cas on y perd une certaine mesure de contrôle en ce sens que certaines *options de contrôle* (sous la forme de sous-langages de $\mathcal{C}_{lo}(L)$) sont inaccessibles à travers une synthèse sur la projection. La synthèse n'est alors pas optimale. Mais dans certaines situations, par exemple l'encapsulation de requis de contrôle de portée exclusivement locale, cette perte d'optimalité de la synthèse peut être acceptable. Le but est généralement d'établir les conditions du théorème 6 de Wong et Wonham [58] nécessaires à la synthèse non bloquante sur l'abstraction.

On définit donc $\mathcal{C}'_{lo} : \mathcal{F}_L \to \mathcal{P}^2(L)$ comme suit

$$\mathcal{C}'_{lo}(H) := \{K \subseteq H \mid (\exists\, K' \in \theta^{-1}(\mathcal{C}_{hi}(M)) \cap \mathcal{F}_L)\ \overline{K} = H \cap K'\} \tag{C.1}$$

et

$$\overline{\mathcal{C}'}_{lo}(H) := \mathcal{C}'_{lo}(H) \cap \mathcal{F}_H \tag{C.2}$$

pour $H \in \mathcal{F}_L$.

Proposition C.1. *Soit $\mathcal{C}_{hi}$ et $\mathcal{C}_{lo}$ deux structures de contrôle standards sur M et L respectivement. Soit $\theta : L \to M$ une projection causale telle que $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$. Alors $\mathcal{C}'_{lo} : \mathcal{F}_L \to \mathcal{P}^2(L)$ telle que définie par l'équation C.1 est une structure de contrôle standard.*

Démonstration. Pour démontrer que $\mathcal{C}'_{lo}$ est une structure de contrôle standard il suffit de vérifier les trois propriétés sur les structures de contrôle standards (voir Wong [55] section 3.1, en particulier la proposition 13) notamment,

(1) $\emptyset, L \in \overline{C'}_{lo}(L)$;

(2) $\overline{C'}_{lo}(L)$ est un sous-treillis complet de $\mathcal{P}(L)$;

(3) $(\forall\, H \in \mathcal{F}_L)\; C'_{lo}(H) = \{K \subseteq H \mid (\exists\, K' \in \overline{C'}_{lo}(L))\; \overline{K} = H \cap K'\}$.

D'abord l'axiome (3) découle directement des définitions, car de l'équation C.1 on a que

$$C'_{lo}(L) = \{K \subseteq L \mid (\exists\, K' \in \theta^{-1}(C_{hi}(M)) \cap \mathcal{F}_L)\; \overline{K} = L \cap K'\}$$

et puisque $K' \subseteq L$ pour tout $K' \in \mathcal{F}_L$, on a

$$\begin{aligned} C'_{lo}(L) &= \{K \subseteq L \mid (\exists\, K' \in \theta^{-1}(C_{hi}(M)) \cap \mathcal{F}_L)\; \overline{K} = K'\} \\ &= \{K \subseteq L \mid \overline{K} \in \theta^{-1}(C_{hi}(M)) \cap \mathcal{F}_L\} \end{aligned}$$

Ce qui nous donne, d'après l'équation C.2 que

$$\begin{aligned} \overline{C'}_{lo}(L) &= C'_{lo}(L) \cap \mathcal{F}_L \\ &= \{K \in C'_{lo}(L) \mid K = \overline{K}\} \\ &= \theta^{-1}(C_{hi}(M)) \cap \mathcal{F}_L \end{aligned}$$

En substituant dans l'équation C.1 on obtient la formulation de l'axiome (3). Il reste donc à démontrer les axiomes (1) et (2).

(1) $\{\emptyset, L\} \subseteq \overline{C'}_{lo}(L) = \theta^{-1}(C_{hi}(M)) \cap \mathcal{F}_L$.

Puisque C_{hi} est une structure de contrôle $\{\emptyset, M\} \subseteq C_{hi}(M) \cap \mathcal{F}_M$. Or puisque $M = \theta(L)$ alors $L \subseteq \theta^{-1}(M)$. Aussi $\theta^{-1}(M) \in \mathcal{F}_L$ (proposition 26 de Wong [55]). Or puisque $K \subseteq L$ pour tout $K \in \mathcal{F}_L$ alors $\theta^{-1}(M) \subseteq L$ et ainsi $L = \theta^{-1}(M)$. Donc $L \in \theta^{-1}(C_{hi}(M)) \cap \mathcal{F}_L$.

L'argument pour $\emptyset$ est trivial.

(2) $\overline{C'}_{lo}(L) = \theta^{-1}(C_{hi}(M)) \cap \mathcal{F}_L$ est un sous-treillis complet de $\mathcal{P}(L)$.

Notez que puisque C_{lo} est une structure de contrôle standard, $\overline{C}_{lo}(L)$ est un sous-

treillis complet de $\mathcal{P}(L)$ (Wong [55] p.35 axiome 2). Or,

$$\begin{aligned} \theta^{-1}(\mathcal{C}_{hi}(M)) &\subseteq \mathcal{C}_{lo}(L) \\ \theta^{-1}(\mathcal{C}_{hi}(M)) \cap \mathcal{F}_L &\subseteq \mathcal{C}_{lo}(L) \cap \mathcal{F}_L = \overline{\mathcal{C}}_{lo}(L) \end{aligned}$$

Prenons $K_i \in \theta^{-1}(\mathcal{C}_{hi}(M)) \cap \mathcal{F}_L$ $i \in \{1,2\}$, Alors $K_i = \theta^{-1}(N_i) \in \overline{\mathcal{C}}_{lo}(L)$ pour un certain $N_i \in \mathcal{C}_{hi}(M)$. Et puisque $K_i \in \mathcal{F}_L$, $N_i = \theta(K_i) = \theta(\overline{K_i}) = \overline{\theta(K_i)} \in \mathcal{F}_M$, on a donc $N_i \in \mathcal{C}_{hi}(M) \cap \mathcal{F}_M = \overline{\mathcal{C}}_{hi}(M)$. Ainsi,

$$K_1 \cup K_2 = \theta^{-1}(N_1) \cup \theta^{-1}(N_2) = \theta^{-1}(N_1 \cup N_2) \in \overline{\mathcal{C}}_{lo}(L)$$

et puisque $N_1 \cup N_2 \in \mathcal{C}_{hi}(M) \cap \mathcal{F}_M$, on a que $K_1 \cup K_2 \in \theta^{-1}(\mathcal{C}_{hi}(M)) \cap \mathcal{F}_L$. De façon similaire,

$$K_1 \cap K_2 = \theta^{-1}(N_1) \cap \theta^{-1}(N_2) = \theta^{-1}(N_1 \cap N_2) \in \overline{\mathcal{C}}_{lo}(L)$$

et puisque $N_1 \cap N_2 \in \mathcal{C}_{hi}(M) \cap \mathcal{F}_M$, on a que $K_1 \cap K_2 \in \theta^{-1}(\mathcal{C}_{hi}(M)) \cap \mathcal{F}_L$.

Ainsi, $\theta^{-1}(\mathcal{C}_{hi}(M)) \cap \mathcal{F}_L \supseteq \{\emptyset, L\}$ et est fermé sous l'union et l'intersection d'ensembles, donc il s'agit d'un sous-trellis complet de L.

On peut donc conclure que $\mathcal{C}'_{lo}$ est une structure de contrôle standard sur L. □

Dans la proposition C.1 on a en fait que $\theta^{-1}(\mathcal{C}_{hi}(M)) \cap \mathcal{F}_L \subseteq \overline{\mathcal{C}}_{lo}(L)$. On a ainsi que $\theta^{-1}(\mathcal{C}_{hi}(M)) \cap \mathcal{F}_L$ constitue aussi un sous-treillis complet de $\overline{\mathcal{C}}_{lo}(L)$ et on peut aisément vérifier que $(\forall H \in \mathcal{F}_L)\ \mathcal{C}'_{lo}(H) \subseteq \mathcal{C}_{lo}(H)$, c'est-à-dire $\mathcal{C}_{lo} \leq \mathcal{C}'_{lo}$.

Corollaire C.2. *Dans les conditions de la proposition C.1, on a $\mathcal{C}_{lo} \leq \mathcal{C}'_{lo}$.*

Démonstration. Puisque, selon la proposition C.1, $\mathcal{C}'_{lo}$ est une structure de contrôle standard, on a que

$$(\forall H \in \mathcal{F}_L)\ \mathcal{C}'_{lo}(H) = \{K \subseteq H \mid (\exists K' \in \overline{\mathcal{C}'}_{lo}(L))\ \overline{K} = H \cap K'\}$$

(axiome 3 sur les structures de contrôle standards).

Or, d'après les conditions de la proposition C.1, $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$ et alors

$$\begin{aligned} \theta^{-1}(\mathcal{C}_{hi}(M)) &\subseteq \mathcal{C}_{lo}(L) \\ \overline{\mathcal{C}'}_{lo}(L) = \theta^{-1}(\mathcal{C}_{hi}(M)) \cap \mathcal{F}_L &\subseteq \mathcal{C}_{lo}(L) \cap \mathcal{F}_L = \overline{\mathcal{C}}_{lo}(L) \end{aligned}$$

d'où $(\forall K \in \mathcal{C}'_{lo}(H))\ K \in \mathcal{C}_{lo}(H)$ puisque $\mathcal{C}_{lo}$ est aussi une structure de contrôle standard. Ainsi $\mathcal{C}'_{lo}(H) \subseteq \mathcal{C}_{lo}(H)$ pour $H \in \mathcal{F}_L$ arbitraire, d'où $\mathcal{C}_{lo} \leq \mathcal{C}'_{lo}$. □

Proposition C.3. *Soit $\mathcal{C}_{hi}$ et $\mathcal{C}_{lo}$ deux structures de contrôle standards sur M et L respectivement. Soit $\theta : L \to M$ une projection causale telle que $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$. Alors*

$$(\forall N \in \mathcal{F}_M)\ \theta(\mathcal{C}'_{lo}(H)) \subseteq \mathcal{C}_{hi}(N)$$

où $H := \theta^{-1}(N)$ et $\mathcal{C}'_{lo} : \mathcal{F}_L \to \mathcal{P}^2(L)$ est telle que définie par l'équation C.1.

Démonstration. Posons $N \in \mathcal{F}_M$, arbitraire, et $H := \theta^{-1}(N)$. Par définition

$$\mathcal{C}'_{lo}(H) = \left\{ K \subseteq H \mid (\exists\, K' \in \overline{\mathcal{C}'}_{lo}(L))\ \overline{K} = H \cap K' \right\}.$$

Or $K' \in \overline{\mathcal{C}'}_{lo}(L) = \theta^{-1}(\mathcal{C}_{hi}(M)) \cap \mathcal{F}_L$, et on a alors que $(\exists\, R' \in \mathcal{C}_{hi}(M))\ K' = \theta^{-1}(R')$. Aussi $\theta(K') = \theta(\theta^{-1}(R')) = R' \in \mathcal{F}_M$ car θ est causale, ainsi $R' \in \overline{\mathcal{C}}_{hi}(M)$.

Maintenant, par la proposition B.3, puisque $H = \theta^{-1}(N)$ on a que $\theta^{-1}(\theta(H)) = H$ et ainsi

$$\begin{aligned} \theta(\overline{K}) = \overline{\theta(K)} &= \theta(H \cap K') \\ &= \theta(H) \cap \theta(K') \\ &= N \cap R' \end{aligned}$$

puisque θ est causale. Mais alors $\theta(K) \subseteq \overline{\theta(K)} \subseteq N$ et $\theta(K) \in \mathcal{C}_{hi}(N)$ puisque $\mathcal{C}_{hi}$ est standard. Ainsi $(\forall N \in \mathcal{F}_M)\ \theta(\mathcal{C}'_{lo}(H)) \subseteq \mathcal{C}_{hi}(N)$ puisque K est arbitraire dans $\mathcal{C}'_{lo}(H)$ et N arbitraire dans $\mathcal{F}_M$. □

Rappelons brièvement que selon la proposition 8 dans [58], pour L et M fermés, lorsque $\theta : L \to M$ est dotée de la propriété d'observateur alors $(\forall N \subseteq M)\ \theta^{-1}(\overline{N}) = \overline{\theta^{-1}(N)}$.

Proposition C.4. *Soit $\mathcal{C}_{hi}$ et $\mathcal{C}_{lo}$ deux structures de contrôle standards sur M et L respectivement. Soit $\theta : L \to M$ une projection causale telle que $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$ et dotée de la propriété d'observateur. Alors*

$$(\forall N \in \mathcal{F}_M)\, \mathcal{C}_{hi}(N) \subseteq \theta(\mathcal{C}'_{lo}(H))$$

où $H := \theta^{-1}(N)$ et $\mathcal{C}'_{lo} : \mathcal{F}_L \to \mathcal{P}^2(L)$ est telle que définie par l'équation C.1.

Démonstration. Posons $N \in \mathcal{F}_M$, arbitraire, et $H := \theta^{-1}(N)$. Puisque $\mathcal{C}_{hi}$ est standard, on a que

$$\mathcal{C}_{hi}(N) = \{R \subseteq N \mid (\exists R' \in \overline{\mathcal{C}}_{hi}(M))\ \overline{R} = N \cap R'\}$$

Maintenant, $R' \in \overline{\mathcal{C}}_{hi}(M) \subseteq \mathcal{C}_{hi}(M)$ et donc $(\exists L' \in \mathcal{C}_{lo}(L))\ L' = \theta^{-1}(R')$ puisque $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$. Aussi, puisque $R' \in \overline{\mathcal{C}}_{hi}(M) \subseteq \mathcal{F}_M$, $\theta^{-1}(R') = L' \in \mathcal{F}_L$ (proposition 26 dans [55]) et donc $L' \in \theta^{-1}(\mathcal{C}_{hi}(M)) \cap \mathcal{F}_L = \overline{\mathcal{C}'}_{lo}(L)$. Or

$$\theta^{-1}(\overline{R}) = \theta^{-1}(N \cap R') = \theta^{-1}(N) \cap \theta^{-1}(R') = H \cap L'$$

et puisque θ est un observateur $\theta^{-1}(\overline{R}) = \overline{\theta^{-1}(R)} = H \cap L'$ et donc $\theta^{-1}(R) \in \mathcal{C}'_{lo}(H)$.

Puisque R est arbitraire dans N et N arbitraire dans $\mathcal{F}_M$, on a $\mathcal{C}_{hi}(N) \subseteq \theta(\mathcal{C}'_{lo}(H))$ pour tout $N \in \mathcal{F}_M$ où $H := \theta^{-1}(N)$. □

On peut en fait établir que, dans les conditions de la proposition C.4, on a plus généralement que $\mathcal{C}_{hi}(N) \subseteq \theta(\mathcal{C}_{lo}(H))$ pour tout $N \in \mathcal{F}_M$ où $H := \theta^{-1}(N)$ (les conditions du théorème 2 de [58]) si l'on note que $\mathcal{C}_{lo} \leq \mathcal{C}'_{lo}$.

Corollaire C.5. *Dans les conditions de la proposition C.4, on a que*

$$(\forall N \in \mathcal{F}_M)\, \mathcal{C}_{hi}(N) \subseteq \theta(\mathcal{C}_{lo}(H))$$

où $H := \theta^{-1}(N)$.

Démonstration. Découle directement de la proposition C.4 et du corollaire C.2. □

Le corollaire C.5 nous assure donc que l'on a la forme affaiblie de la consistance du

contrôle ([58], théorème 2) pour le triplet $(L, \theta, \mathcal{C}_{lo})$ notamment :

$$\mathcal{C}_{hi}(H) \subseteq \theta(\mathcal{C}_{lo}(N)) \iff \theta \circ \kappa_N \circ \theta^{-1}|_{\mathcal{C}_{hi}(H)} = id_{\mathcal{C}_{hi}(H)} \tag{C.3}$$

pour $N \in \mathcal{F}_M$ et $H := \theta^{-1}(N)$.

Finalement, des propositions C.3 et C.4, on peut conclure à la consistance du contrôle complète (les conditions du théorème 1 de [58]) pour le triplet $(L, \theta, \mathcal{C}'_{lo})$.

Théorème C.6. *Soit $\mathcal{C}_{hi}$ et $\mathcal{C}_{lo}$ des structures de contrôle standards sur M et L respectivement. Soit $\theta : L \to M$ une projection causale telle que $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$ et dotée de la propriété d'observateur. Alors*

$$(\forall N \in \mathcal{F}_M)\ \mathcal{C}_{hi}(N) = \theta(\mathcal{C}'_{lo}(H))$$

où $H := \theta^{-1}(N)$ et $\mathcal{C}'_{lo} : \mathcal{F}_L \to \mathcal{P}^2(L)$ est telle que définie par l'équation C.1.

Démonstration. Découle directement des propositions C.3 et C.4. □

Proposition C.7. *Soit $\mathcal{C}_{hi}$ et $\mathcal{C}_{lo}$ des structures de contrôle standards sur M et L respectivement. Soit $\theta : L \to M$ une projection causale telle que $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$ et dotée de la propriété d'observateur. Alors*

$$(\forall E \subseteq M)\ \kappa'_L \circ \theta^{-1}(E) = \theta^{-1}(\kappa_M(E))$$

où κ'_L est l'opérateur de contrôle associé à $\mathcal{C}'_{lo} : \mathcal{F}_L \to \mathcal{P}^2(L)$ telle que définie par l'équation C.1.

Démonstration. ($\subseteq$) D'une part, par le théorème C.6, on a $\mathcal{C}_{hi}(M) = \theta(\mathcal{C}'_{lo}(L))$ et donc $\kappa_M = \theta \circ \kappa'_L \circ \theta^{-1}$ par consistance du contrôle. Ainsi on a que

$$\begin{aligned} \theta^{-1}(\theta \circ \kappa'_L \circ \theta^{-1}(E)) &= \theta^{-1}(\kappa_M(E)) \\ \theta^{-1}(\theta(\kappa'_L \circ \theta^{-1}(E))) &= \theta^{-1}(\kappa_M(E)). \end{aligned}$$

Et puisque $\kappa'_L \circ \theta^{-1}(E) \subseteq \theta^{-1}(\theta(\kappa'_L \circ \theta^{-1}(E)))$ en général, on a

$$\kappa'_L \circ \theta^{-1}(E) \subseteq \theta^{-1}(\theta(\kappa'_L \circ \theta^{-1}(E))) = \theta^{-1}(\kappa_M(E)).$$

($\supseteq$) D'autre part on a que

$$\begin{aligned} E &\supseteq \kappa_M(E) \\ \theta^{-1}(E) &\supseteq \theta^{-1}(\kappa_M(E)) \\ \kappa'_L \circ \theta^{-1}(E) &\supseteq \kappa'_L \circ \theta^{-1}(\kappa_M(E)) = \theta^{-1}(\kappa_M(E)) \end{aligned}$$

puisque $\theta^{-1}(\kappa_M(E)) \subseteq \mathcal{C}'_{lo}(L)$ par définition de $\mathcal{C}'_{lo}(L)$.

□

À ce stade les conditions pour assurer la consistance du contrôle ont été établies (voir la proposition C.1 et le théorème C.6). Il suffit que $\mathcal{C}_{hi}$ et $\mathcal{C}_{lo}$ soient des structures de contrôle standards sur M et L respectivement, et que $\theta : L \to M$ soit une projection causale telle que $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$ et dotée de la propriété d'observateur. On peut alors conclure que $\mathcal{C}'_{lo} : \mathcal{F}_L \to \mathcal{P}(L)$ est une structure de contrôle standard et que pour $N \in \mathcal{F}_M$ et $H := \theta^{-1}(N)$ on a $\mathcal{C}_{hi}(N) = \theta(\mathcal{C}'_{lo}(H))$. On a en particulier, d'après le théorème 1 dans [58], que $\mathcal{C}_{hi}(M) = \theta(\mathcal{C}'_{lo}(L)) \subseteq \theta(\mathcal{C}_{lo}(L))$ et donc $\kappa_M = \theta \circ \kappa'_L \circ \theta^{-1}$ où κ'_L est l'opérateur de synthèse (*control operator*) associé à $\mathcal{C}'_{lo}$. Toutefois $\mathcal{C}'_{lo}$ n'est pas localement définissable et ne dispose donc pas d'une technologie de contrôle.

C.2 Synthèse non bloquante

La proposition suivante est établie d'après une argumentation qu'on peut trouver dans [59] (en page 278). Le lecteur peut se référer à la section 2.3.2.1 pour les définitions de L_{loc}, θ_v et θ_v^{-1}, et à la section 2.3.1.2 pour la définition de $\overline{\kappa}_M$.

Proposition C.8. *Soit $\mathcal{C}_{hi}$ et $\mathcal{C}_{lo}$ des structures de contrôle standards sur M et L respectivement. Soit $\theta : L \to M$ une projection causale telle que $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$ et dotée de la propriété d'observateur. Alors*

$$\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$$

c'est-à-dire que θ est exempte de couplage de contrôle (voir section B.2).

Démonstration.
Il est clair que pour tout $E \in \mathcal{F}_M$ on a certainement que $\overline{\kappa}_M(E) = \kappa_M(E) \in \mathcal{C}_{hi}(M)$. Mais alors,

$$\theta_v^{-1}(\overline{\kappa}_M(E)) \subseteq \theta^{-1}(\overline{\kappa}_M(E)) = \kappa_L(\theta^{-1}(\overline{\kappa}_M(E)))$$

puisque $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$, et donc $\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa_L \circ \theta^{-1} \circ \overline{\kappa}_M$ pour tout $E \in \overline{\mathcal{C}}_{hi}(M)$. Ainsi θ est exempte de couplage de contrôle. □

Corollaire C.9. *Dans les conditions de la proposition C.8 on a aussi que*

$$\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa'_L \circ \theta^{-1} \circ \overline{\kappa}_M$$

où κ'_L est l'opérateur de contrôle associé à $\mathcal{C}'_{lo}$: $\mathcal{F}_L \rightarrow \mathcal{P}^2(L)$ telle que définie par l'équation C.1.

Démonstration. L'argument est essentiellement le même que pour la proposition C.8 puisque $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}'_{lo}(L)$ par définition de $\mathcal{C}'_{lo}(L)$. □

En somme, les conditions pour une synthèse non bloquante sur l'abstraction sont maintenant presque toutes réunies (théorème 6 de [58]). Il ne reste plus qu'à obtenir la condition

$$\theta^{-1}(M_m) = L_m \,. \tag{C.4}$$

Cette condition n'est présente dans le théorème 6 que pour assurer la *consistance du marquage* entre les deux niveaux et ainsi pouvoir établir un lien théorique clair entre une solution non bloquante dans l'*abstraction* et son implémentation non bloquante dans l'*agent*. Comme mentionné dans [60] en préambule : «*cette condition est aisée à produire localement*». En fait on la pose tout simplement comme une définition $L'_m := \theta^{-1}(M_m)$. Ce faisant on élargit en général L_m mais, avec la propriété d'observateur, on sait que lorsque l'abstraction a produit $t \in M_m$ alors l'agent est en portée *silencieuse* d'au moins une chaîne de L_m. D'un point de vue pragmatique, cette assurance suffit puisque le système est toujours capable de déterminer si la configuration courante est terminale ou non ; l'abstraction ne joue aucun rôle ici.

Théorème C.10. *Soit C_{hi} et C_{lo} des structures de contrôle standards sur M et L respectivement. Soit $\theta : L \to M$ une projection causale telle que $\theta^{-1}(C_{hi}(M)) \subseteq C_{lo}(L)$ et dotée de la propriété d'observateur. Si on pose $\theta^{-1}(M_m) = L_m$, alors*

$$(\forall E \subseteq M)\, \kappa_M(E) \textit{ non bloquant} \iff \theta^{-1}(\kappa_M(E)) \textit{ non bloquant.}$$

Démonstration. Sous les conditions du théorème on sait que $C'_{lo} : \mathcal{F}_L \to \mathcal{P}^2(L)$, telle que définie par l'équation C.1, existe et qu'il s'agit d'une structure de contrôle standard (proposition C.1). On sait que la restriction de l'équation 2.6 est toujours réalisée pour les structures de contrôle standards. Aussi, par le théorème C.6, on a la consistance du contrôle pour le triplet (L, θ, C'_{lo}) de sorte que $C_{hi}(M) = \theta(C'_{lo}(L))$. On a enfin que

- $\theta^{-1}(M_m) = L_m$ par hypothèse ;
- θ est dotée de la propriété d'observateur par hypothèse ;
- $\theta_v^{-1} \circ \overline{\kappa}_M \leq \kappa'_L \circ \theta^{-1} \circ \overline{\kappa}_M$ (corollaire C.9) ;

où κ'_L est l'opérateur de contrôle associé à C'_{lo}. Ainsi le théorème 6 de [58] est applicable dans cette situation et on a

$$(\forall E \subseteq M)\, \kappa_M(E) \text{ non bloquant} \iff \kappa'_L \circ \theta^{-1}(E) \text{ non bloquant.}$$

D'où on peut conclure, par la proposition C.7, que

$$(\forall E \subseteq M)\, \kappa_M(E) \text{ non bloquant} \iff \theta^{-1}(\kappa_M(E)) \text{ non bloquant.}$$

□

Reste à établir qu'il est possible de construire θ pour que les conditions précitées existent.

C.3 Règles de construction de la projection

Supposons $L = \overline{L_m} \subseteq \Sigma^*$ le comportement fermé d'un *agent*, muni d'une technologie de contrôle standard $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$. Alors L est muni d'une structure de contrôle standard localement définissable. Supposons une projection causale $\theta : L \to M$ dotée de la propriété

d'observateur, l'expression d'une *abstraction* sur l'agent. Puisque L est préfixe clos, alors $M = \theta(L)$ est préfixe clos. De plus le comportement terminal de l'abstraction est alors donné par $M_m := \theta(L_m)$.

Rappelons brièvement que

$$L_{voc} := \{\varepsilon\} \cup \omega^{-1}(T) \subseteq L$$

où $\omega : L \to T$ est une fonction partielle appelée fonction de sortie de la projection θ. La projection θ peut alors être définie comme suit :

$$\begin{aligned} \theta(\varepsilon) &= \varepsilon \\ \theta(s\sigma) &= \begin{cases} \theta(s)\omega(s\sigma), & \text{si } \omega(s\sigma)! \text{ et } \omega(s\sigma) = \tau \in T \\ \theta(s) & \textit{autrement} \end{cases} \end{aligned}$$

où $\omega(s\sigma)!$ signifie que $\omega(s\sigma)$ est définie. Alors L_{voc} forme l'ensemble des chaînes de L provoquant la génération d'un événement à travers θ. Aussi θ_v est la restriction de θ à L_{voc} et θ_v^{-1} la fonction θ^{-1} dont le co-domaine est restreint à L_{voc} (voir section 2.3.2.1).

Définissons maintenant

$$\begin{aligned} X_\tau &:= \{s\sigma \in L_{voc} \mid \omega(s\sigma) = \tau\} \\ T_\theta &:= \{\tau \in T \mid X_\tau \neq \emptyset\} \\ T_c &\subseteq \{\tau \in T_\theta \mid X_\tau \subseteq \Sigma^*\Sigma_c\} \\ T_u &:= T_\theta - T_c \,. \end{aligned} \tag{C.5}$$

Sans perte de généralité on peut supposer que $T = T_\theta$. On a donc $T = T_c \dot{\cup} T_u$, et M est alors doté d'une technologie de contrôle standard induisant $\mathcal{C}_{hi}$ une structure de contrôle standard localement définissable dans l'abstraction. Clairement, pour $l\tau \in M$ telle que $\tau \in T_c$, on a $\theta_v^{-1}(l\tau) = \theta^{-1}(l\tau) \cap L_{voc} \subseteq \Sigma^*\Sigma_c$ et plus généralement on a

$$(\forall \tau \in T_c)\ \emptyset \neq \omega^{-1}(\tau) \subseteq \Sigma^*\Sigma_c. \tag{C.6}$$

La formule C.6 caractérise l'absence de délai de contrôle de la projection. Une caractérisa-

tion différente mais équivalente est donnée dans [59]. La proposition qui suit montre que l'absence de délai de contrôle entraîne la coïncidence du contrôle (section B.3).

Proposition C.11. *Soit $L \subseteq \Sigma^*$ muni d'une technologie de contrôle standard c'est-à-dire $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$. Soit $M \in T^*$ et $\theta : L \to M$ une projection causale dotée de la propriété d'observateur, avec T et θ conçus en respectant les règles de équation C.5, de sorte que M soit muni d'une technologie de contrôle standard $T = T_c \dot{\cup} T_u$ où la formule C.6 est vérifiée. Alors $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$.*

Démonstration. (Argument similaire à celui du lemme 17 de [59], appendice A)
Posons $K \in \theta^{-1}(\mathcal{C}_{hi}(M))$, alors $K = \theta^{-1}(N)$ pour un certain $N \in \mathcal{C}_{hi}(M)$. Puisque $N \subseteq M = \theta(L)$, on a que $\theta(K) = \theta(\theta^{-1}(N)) = N$. On doit démontrer que $K \in \mathcal{C}_{lo}(L)$. Posons donc $s\sigma \in \overline{K}\Sigma_u \cap L$, c'est-à-dire $s \in \overline{K}$, $\sigma \in \Sigma_u$ et $s\sigma \in L$. Il y a deux cas.

(1) $\theta(s\sigma) = \theta(s)$: Alors, $\theta(s\sigma) = \theta(s) \in \theta(\overline{K}) = \overline{\theta(K)} = \overline{N}$, puisque θ est causale. Donc $s\sigma \in \theta^{-1}(\overline{N}) = \overline{\theta^{-1}(N)} = \overline{K}$, puisque θ est dotée de la propriété d'observateur.

(2) $\theta(s\sigma) = \theta(s)\tau$: Alors $s\sigma \in \omega^{-1}(\tau)$ et puisque $\sigma \in \Sigma_u$, par hypothèse $\tau \in T_u$. Or $\theta(s) \in \theta(\overline{K}) = \overline{\theta(K)} = \overline{N}$, et $\theta(s\sigma) \in \theta(L) = M$, donc puisque N est contrôlable, $\theta(s\sigma) = \theta(s)\tau \in \overline{N}T_u \cap M \subseteq \overline{N}$. Ainsi, encore une fois $s\sigma \in \theta^{-1}(\overline{N}) = \overline{\theta^{-1}(N)} = \overline{K}$ puisque θ est dotée de la propriété d'observateur.

□

Corollaire C.12. *Dans les conditions de la proposition C.11, en supposant la consistance du marquage ($\theta^{-1}(\theta(L_m)) = L_m$), alors*

$$(\forall\, E \subseteq M)\, \kappa_M(E) \textit{ non bloquant} \iff \theta^{-1}(\kappa_M(E)) \textit{ non bloquant.}$$

Démonstration. Le résultat découle directement de ce que $\mathcal{C}_{hi}$ et $\mathcal{C}_{lo}$ sont toutes deux munies de technologies de contrôle et sont par conséquent des structures de contrôle standards. On retrouve ainsi les conditions du théorème C.10. □

Dans les conditions de la proposition C.11, en supposant la consistance du marquage ($\theta^{-1}(\theta(L_m)) = L_m$), pour un objectif de contrôle $E \subseteq M$ alors $N := \kappa_M(E)$ est contrôlable et non bloquant par rapport à M et $K := \theta^{-1}(N)$ est contrôlable et non bloquant

par rapport à L (théorème C.10). De toute évidence $\theta(K) = N$ et en supposant $N \neq \emptyset$ M_m-fermé, on a aussi que $K = \theta^{-1}(N) \subseteq \theta^{-1}(\theta(L_m)) = L_m$ et K est L_m-fermé. Dans ces circonstances, selon la proposition 85 dans [55], il existe deux lois de contrôle $V_N : M \to \mathcal{P}(T)$ et $V_K : L \to \mathcal{P}(\Sigma)$ telles que $M(V_N) = \overline{N}$ et $L(V_K) = \overline{K}$.

Pour pouvoir *implémenter* le résultat d'une synthèse sur l'*abstraction* ($\mathbf{G}_{hi}$) dans l'*agent* ($\mathbf{G}_{lo}$) il faut pouvoir lier V_N et V_K par une règle telle que celle de l'équation 2.7. En d'autres termes in faut que V_N et V_K soient *synchrones* (section B.4). Or, dans les conditions de la proposition C.11, il manque un élément à savoir que la structure de contrôle $\mathcal{C}'_{lo}$ doit être *localement définissable*. Or, à cause de la règle de définition de la projection (équation C.5), bien que $\mathcal{C}'_{lo}$ soit standard elle n'est justement pas *localement définissable*. Mais on se rappelle que $\mathcal{C}_{lo} \leq \mathcal{C}'_{lo}$ (corollaire C.2) et qu'en conséquence $\mathcal{C}_{hi}(N) \subseteq \theta(\mathcal{C}_{lo}(H))$ (pour $N \in \mathcal{F}_M$ et $H := \theta^{-1}(N)$, corollaire C.5) ce qui correspond à la forme affaiblie de la consistance du contrôle. Tel que mentionné en annexe A, les résultats sur le contrôle non bloquant restent valides avec cette forme de consistance du contrôle. Les lois de contrôle V_N et V_K sont donc synchrones puisque θ est un observateur et qu'il y a absence de couplage de contrôle (proposition C.8). Dans ce cas, puisqu'on a que $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$ (conditions de la proposition C.11), la règle de l'équation 2.7 se simplifie de la façon suivante :

$$V_K(s) = \Sigma_u \cup \{\sigma \in \Sigma_c \mid (\forall \tau \in T)\ \theta(s\sigma) = \theta(s)\tau \implies \tau \in V_N(\theta(s))\} \qquad \text{(C.7)}$$

pour $s \in L$. On se propose donc d'implémenter V_K à l'aide de la technologie de contrôle de $\mathcal{C}_{lo}$ ($\Sigma = \Sigma_c \dot{\cup} \Sigma_u$ la technologie de contrôle standard), ce qui semble raisonnable puisque $\mathcal{C}_{lo} \leq \mathcal{C}'_{lo}$.

Proposition C.13. *Dans les conditions de la proposition C.11,*

$$L(V_K) = \theta^{-1}(\overline{\kappa_M(E)})$$

où V_K est définie par l'équation C.7.

Démonstration.
Observons d'abord que $\kappa_M(E) \in \mathcal{C}_{hi}(M)$ par définition et qu'ainsi $\overline{\kappa_M(E)} \in \mathcal{C}_{hi}(M)$

puisque $\mathcal{C}_{hi}$ est une structure de contrôle. Or, puisque $\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L)$, on a que $\theta^{-1}(\overline{\kappa_M(E)}) \in \mathcal{C}_{lo}(L)$. D'autre part $\theta^{-1}(\overline{\kappa_M(E)}) \in \mathcal{F}_L$ ([58], proposition 4). Maintenant posons $\kappa_M(E) \neq \emptyset$ de sorte que $\theta^{-1}(\overline{\kappa_M(E)}) \neq \emptyset$ et ainsi $\varepsilon \in L(V_K) \cap \theta^{-1}(\overline{\kappa_M(E)})$.

($\subseteq$) Posons $s\sigma \in L(V_K)$ c'est-à-dire $s \in L(V_K)$, $\sigma \in V_K(s)$ et $s\sigma \in L$, et faisons l'hypothèse d'induction que $s \in \theta^{-1}(\overline{\kappa_M(E)})$. Alors,

(1) ou bien $\sigma \in \Sigma_u$;

(2) ou bien $\sigma \in \Sigma_c$ et $\theta(s\sigma) = \theta(s)$;

(3) ou bien $\sigma \in \Sigma_c$, $\theta(s\sigma) = \theta(s)\tau$ et $\tau \in V_N(\theta(s))$.

Dans le cas (1), $s\sigma \in \theta^{-1}(\overline{\kappa_M(E)})\Sigma_u \cap L \subseteq \theta^{-1}(\overline{\kappa_M(E)})$, puisque $\theta^{-1}(\overline{\kappa_M(E)})$ est contrôlable. Dans le cas (2), $\theta(s\sigma) = \theta(s) \in \overline{\kappa_M(E)}$, puisque par hypothèse $s \in \theta^{-1}(\overline{\kappa_M(E)})$, et ainsi $s\sigma \in \theta^{-1}(\overline{\kappa_M(E)})$. Dans le cas (3), puisque par hypothèse $s \in \theta^{-1}(\overline{\kappa_M(E)})$, on a $\theta(s) \in \overline{\kappa_M(E)} = M(V_N)$. Mais alors, puisque $s\sigma \in L$, on a $\theta(s)\tau = \theta(s\sigma) \in \theta(L) = M$, d'où $\theta(s\sigma) = \theta(s)\tau \in M(V_N) = \overline{\kappa_M(E)}$ et donc $s\sigma \in \theta^{-1}(\overline{\kappa_M(E)})$.

($\supseteq$) Posons $s\sigma \in \theta^{-1}(\overline{\kappa_M(E)})$, on a donc que $\theta(s\sigma) \in \overline{\kappa_M(E)} \subseteq M$, et faisons l'hypothèse d'induction que $s \in L(V_K)$. Tentons de supposer pour un instant que $s\sigma \notin L(V_K)$. Alors, par définition de $L(V_K)$ (section 2.3.1.4), $\sigma \notin V_K(s)$ et, par l'équation C.7, $\sigma \in \Sigma_c$, $\theta(s\sigma) = \theta(s)\tau$ et $\tau \notin V_N(\theta(s))$. Mais alors on obtient que $\theta(s\sigma) = \theta(s)\tau \notin M(V_N) = \overline{\kappa_M(E)}$ ce qui contredit l'hypothèse selon laquelle $\theta(s\sigma) \in \overline{\kappa_M(E)} \subseteq M$ (c'est-à-dire $s\sigma \in \theta^{-1}(\overline{\kappa_M(E)})$). On en conclut donc que $s\sigma \in L(V_K)$. □

Cette preuve est très similaire à celle qu'on retrouve pour la proposition 1 dans [57]. En effet le genre de projection engendrée par l'équation C.5 s'apparente à une projection trouée. On note aussi que les projections naturelles sont un cas particulier de projections trouées. Or dans ces deux cas il est évident que, par définition, ces deux types de projections sont exemptes de délai de contrôle (formule C.6) puisqu'elles exposent (ou effacent) *directement* certaines transitions sur événements contrôlables de leur dynamique, et que leur alphabet est alors un *sous-ensemble* de l'alphabet original. Si en plus on dote ces projections de la propriété d'observateur (par exemple comme dans [14] pour les *observateurs*

naturels et dans [57] pour les projections trouées), on obtient des résultats similaires au résultat précédent.

C.4 Composition synchrone d'abstractions

L'assemblage de *composantes* pour produire le modèle de comportement libre d'un sous-système se fait généralement par produit synchrone dans le contexte usuel de SCT. Il est intéressant de constater que l'absence de délai de contrôle se propage au produit synchrone de deux (ou plusieurs) abstractions obtenues par la règle de l'équation C.5.

Soit $\Sigma_i := \Sigma_{i,c} \dot{\cup} \Sigma_{i,u}$ $(i = 1, 2)$ deux alphabets disjoints et $\Sigma := \Sigma_1 \dot{\cup} \Sigma_2$. Soit aussi $q_i : \Sigma^* \to \Sigma_i$ $(i = 1, 2)$ les projections naturelles correspondantes. Soit $L_i \subseteq \Sigma_i^*$ $(i = 1, 2)$ des langages préfixes clos et $L := L_1 \parallel L_2$. Soit $T_i := T_{i,c} \dot{\cup} T_{i,u}$ $(i = 1.2)$ deux alphabets disjoints, $T := T_1 \dot{\cup} T_2$ et $P_i : T^* \to T_i^*$ $(i = 1, 2)$ les projections naturelles correspondantes. Soit $\theta_i : L_i \to T_i^*$ $(i = 1, 2)$ des projections causales définies comme suit

$$\begin{aligned} \theta_i(\varepsilon) &= \varepsilon \\ \theta_i(s\sigma) &= \begin{cases} \theta_i(s) \cdot \omega_i(s\sigma), & \text{si } \omega_i(s\sigma)! \\ \theta_i(s), & \text{autrement} \end{cases} \end{aligned}$$

pour $s\sigma \in L_i$ et $\omega_i : L_i \to T_i$ une fonction partielle arbitraire appelée fonction de sortie de θ_i. On suppose aussi que, pour $i \in \{1.2\}$, $\Sigma_i \cap T_i = \emptyset$ et que (L_i, θ_i, T_i) ont été conçus conformément à la règle de l'équation C.5. On a donc que la formule C.6 est vérifiée pour (L_i, θ_i, T_i) $i = 1.2$. On peut alors former $\omega : L \to T$ comme suit

$$\omega(s\sigma) := \begin{cases} \omega_i(q_i(s\sigma)), & \text{si } \sigma \in \Sigma_i \text{ et } \omega_i(q_i(s\sigma))! \ (i = 1, 2) \\ \textit{indéfinie}, & \text{autrement} \end{cases}$$

pour $s\sigma \in L$. Dans cette définition, on note que $\omega(s\sigma)$ est bien définie puisque Σ_1 et Σ_2

sont disjoints. Avec ω ainsi constituée on peut former $\theta : L \to T^*$ de la façon suivante

$$\begin{aligned} \theta(\varepsilon) &= \varepsilon \\ \theta(s\sigma) &= \begin{cases} \theta(s) \cdot \omega(s\sigma), & \text{si } \omega(s\sigma)! \\ \theta(s), & \text{autrement} \end{cases} \end{aligned}$$

pour $s\sigma \in L$. D'après Pu [42], dans les conditions précitées, cette définition de θ constitue la synchronisation de θ_1 et θ_2 notée $\theta := \theta_1 \parallel \theta_2$ et on a alors :

$$\begin{aligned} P_i \circ \theta &= \theta_i \circ q_i \quad i = 1, 2 \\ q_i \circ \theta^{-1} &= \theta_i^{-1} \circ P_i \quad i = 1, 2 \\ \theta(L) &= \theta_1(L_1) \parallel \theta_2(L_2) \end{aligned}$$

correspondant respectivement au lemme A.7, au lemme A.8 et à la proposition A.5 dans [42].

Sans perte de généralité on peut poser $M := \theta(L)$ et restreindre le co-domaine de θ à M pour donner $\theta : L \to M$, et de la même façon obtenir $\theta_i : L_i \to M_i$ $(i = 1, 2)$. On appelle alors les systèmes (L_i, M_i, θ_i) des composantes disjointes du système (L, M, θ).

Proposition C.14. *Pour un système (L, M, θ), obtenu par produit synchrone de deux composantes disjointes (L_i, M_i, θ_i) telles que définies précédemment, alors la formule C.6 est vérifiée pour (L, M, θ).*

Démonstration.
Prenons $\tau \in T_c$ arbitraire. Alors, puisque $T_1 \cap T_2 = \emptyset$, ou bien $\tau \in T_{1,c}$ ou bien $\tau \in T_{2,c}$. Supposons donc que $\tau \in T_{1,c}$. Prenons maintenant $s\sigma \in \omega^{-1}(\tau)$. Puisque $\omega^{-1}(\tau) \subseteq L_{voc}$ par définition et que $s < s\sigma$, on a donc $s \in L \subseteq \Sigma^*$. Mais alors, par définition on a que $\omega_2(q_2(s)\sigma)$ n'est pas définie, $\tau = \omega(s\sigma) = \omega_1(q_1(s)\sigma)$ et $\sigma \in \Sigma_{1,c} \subseteq \Sigma_c$. Ainsi $s\sigma \in \Sigma^*\Sigma_c$. On peut faire un argument symétrique avec la même conclusion pour $\tau \in T_{2,c}$. Puisque τ est arbitraire dans T_c , on en conclut que la formule C.6 est vérifiée pour (L, M, θ). □

Puisque le produit synchrone est associatif, ce résultat se généralise à un nombre fini, arbitraire de composantes.

Une question intéressante est de savoir ce qui arrive à la propriété d'observateur lorsque l'on opère la composition synchrone de deux modèles qui sont eux-mêmes des abstractions (θ_i pour $i = 1, 2$) possédant cette propriété. On trouve une réponse à cette question dans Pu [42] (aux sections 2.3 et A.2) et aussi une réponse partielle, concernant les observateurs naturels (projections naturelles possédant la propriété d'observateur), dans Wong et Wonham [59] (proposition 8). Le contexte dans [59] étant plus près de celui utilisé dans la présente étude, il est utile d'utiliser la proposition 8 et de la généraliser pour des projections causales (en utilisant certains résultats de [42]).

Proposition C.15. *Soit un système (L, M, θ) obtenu par composition synchrone de deux composantes (L_i, M_i, θ_i), $i = 1, 2$, disjointes définies comme dans ce qui précède. Supposons que pour $i = 1, 2$, les projections θ_i sont dotées de la propriété d'observateur. Alors la projection $\theta := \theta_1 \parallel \theta_2$ est aussi dotée de la propriété d'observateur.*

Démonstration.
On observe d'abord que d'après le lemme A.7 dans [42] on a

$$P_i \circ \theta = \theta_i \circ q_i \tag{C.8}$$

pour θ, θ_i, P_i et q_i ($i = 1, 2$) définis comme précédemment.

Posons $s \in L$ et $\tau \in T$ tels que $\theta(s)\tau \in \theta(L) = M$. Puisque $\Sigma_1 \cap \Sigma_2 = \emptyset$ on a $q_i(s) \in q_i(L) = L_i$ pour $i = 1, 2$. Supposons que $\tau \in T_1$. Alors par l'équation C.8 on a

$$\begin{aligned} & \theta(s)\tau \in \theta(L) \\ \Longrightarrow \quad & P_1(\theta(s)\tau) \in P_1(\theta(L)) \\ \Longrightarrow \quad & (P_1(\theta(s))\tau) \in \theta_1(q_1(L)) \\ \Longrightarrow \quad & \theta_1(q_1(s))\tau \in \theta_1(L_1)\,. \end{aligned}$$

Or puisque θ_1 est un observateur, il existe donc $u \in \Sigma_1^*$ pour laquelle $q_1(s)u \in L_1$ et $\theta_1(q_1(s)u) = \theta_1(q_1(s))\tau$ ([58], proposition 7). Sans perte de généralité on peut considérer que $q_1(s)u \in L_{1.voc}$.

Puisque $u \in \Sigma_1^*$ alors $q_1(su) = q_1(s)u \in L_1$ donc $su \in {q_1}^{-1}(L_1)$. Aussi, puisque

$\Sigma_1 \cap \Sigma_2 = \emptyset$, on a $q_2(su) = q_2(s) \in L_2$ et donc $su \in q_2^{-1}(L_2)$. Ainsi

$$su \in q_1^{-1}(L_1) \cap q_2^{-1}(L_2) = L$$

et $\theta(su) \in \theta(L) = M$. Or $\theta(s) \leq \theta(su)$ puisque θ est causale, et donc $\theta(su) = \theta(s)t$ pour un certain $t \in T^*$. Maintenant puisque $q_1(s)u \in L_{1,uoc}$ on a que $su \in L_{uoc}$ ([42], lemme A.9) d'où $t \in T^+$. Mais puisque $\theta_2(q_2(su)) = \theta_2(q_2(s))$, on a que $t \in T_1^+$. Et puisque $\theta_1(q_1(su)) = \theta_1(q_1(s)u) = \theta_1(q_1(s))\tau$, on a finalement que $t = \tau$ et ainsi $\theta(su) = \theta(s)\tau$.

Par symétrie on peut suivre le même argument en supposant $\tau \in T_2$. Ainsi, par les proposition 7 et 8 dans [58], on peut conclure que la projection $\theta = \theta_1 \parallel \theta_2$ est aussi dotée de la propriété d'observateur. □

Puisque le produit synchrone est associatif il est possible de généraliser ce résultat à un nombre fini, arbitraire de composantes disjointes.

Proposition C.16. *Soit un système (L, M, θ) obtenu par composition synchrone de deux composantes (L_i, M_i, θ_i), $i = 1, 2$, disjointes définies comme dans ce qui précède. Supposons que pour $i = 1, 2$, les projections θ_i sont dotées de la propriété d'observateur. Alors la projection $\theta := \theta_1 \parallel \theta_2$ est dotée de la propriété de* coïncidence du contrôle *définie en section B.3, notamment*

$$\theta^{-1}(\mathcal{C}_{hi}(M)) \subseteq \mathcal{C}_{lo}(L) .$$

Démonstration.

Premièrement, pour le système de bas niveau $L = L_1 \parallel L_2$, on peut toujours associer des générateurs $\mathbf{G}_i$ où $L_i = L(\mathbf{G}_i)$ $(i = 1, 2)$. La relation entre produit synchrone de langages et produit synchrone de générateurs (équation 1.3) nous assure alors que

$$L = L(\mathbf{G}_1 \parallel \mathbf{G}_2) = L(\mathbf{G}_1) \parallel L(\mathbf{G}_1) = L_1 \parallel L_2 .$$

Ainsi, d'après la définition de produit synchrone de générateurs (équation 1.2), il devrait être évident que si L_i $(i = 1, 2)$ sont munis d'une technologie de contrôle standard, alors L est aussi muni d'une telle technologie de contrôle. D'ailleurs, cette manoeuvre est typique du *contexte usuel* de SCT (contexte de Ramadge et Wonham).

Deuxièmement, θ_i ($i = 1, 2$) sont des projections causales dotées de la propriété d'observateur et construites d'après les règles de l'équation C.5. Alors $M_i = \theta_i(L_i)$ ($i = 1, 2$) sont aussi dotés d'une technologie de contrôle standard et, par un argument similaire au précédent, M est donc aussi doté d'une technologie de contrôle standard.

Troisièmement, les projections θ_i ($i = 1, 2$) vérifient toutes deux la formule C.6, donc par la proposition C.14 la projection $\theta := \theta_1 \parallel \theta_2$ vérifie aussi la formule C.6.

Finalement, puisque $\theta := \theta_1 \parallel \theta_2$ est aussi dotée de la propriété d'observateur, d'après la proposition C.15, alors la proposition C.11, nous assure le résultat escompté. □

On peut donc obtenir une représentation *abstraite* d'un système par combinaison d'un ensemble de représentations *abstraites* de ses *composantes*, pourvu que ces dernières soient disjointes entre elles. Si l'ensemble des composantes abstraites sont des observateurs (des composantes concrètes qu'elles représentent) le modèle obtenu par composition synchrone de ces abstractions est aussi un observateur du système concret (c'est-à-dire une *abstraction* de la composition synchrone des composantes *concrètes*). Au sens de Pu [42] il s'agit *d'abstractions fiables* (de l'anglais *reliable abstraction*, section 2.7). Cette situation n'est pas sans rappeler la façon dont un système est modélisé dans le contexte usuel de SCT (modèle de Ramadge et Wonham).

Annexe D

Inventaire de composantes réutilisables

Cette section présume que le lecteur est familier avec le contenu du chapitre 4. Elle décrit l'abstraction en composantes réutilisables des dispositifs disponibles sur l'usine école *MPS* (*Modular Production System*) de ***Festo***^{TM}, dont on peut trouver la description détaillée dans l'ensemble des brochures [17] à [21]. La bibliothèque de composantes réutilisables est divisée en deux parties :

1. l'abstraction des dispositifs physiques de l'usine *MPS* (composantes élémentaires) ;
2. une collection de sous-systèmes abstraits formant des unités utiles pour la description de problèmes de contrôle sur l'usine école (composantes mixtes).

L'ensemble est orienté vers la solution d'une étude de cas décrite au chapitre 5. On trouve à la fin de cette section quelques statistiques sur la complexité de l'ensemble de ces composantes.

Dans ce qui suit, lors de l'élaboration de spécifications de contrôle, on classifie l'ensemble des requis de contrôle en trois groupes : requis *implicites*, *explicites* ou *standards*. Cette classification n'apparaît nulle part (comme telle) dans la littérature sur SCT et mérite d'être clarifiée.

Selon la méthode préconisée en section 4.1, on décrit un sous-système par un modèle de STS sans structure verticale. De façon *standard*, avec des modèles de STS, on modélise les contraintes de contrôle par usage de modèles pour la mémoire. Ces modèles sont *fermés* sur l'ensemble de leurs événements incontrôlables (habituellement par des transitions en boucle sur le même état d'où elles émanent), et toute contrainte de contrôle visant à exclure

une ou plusieurs de ces transitions doit être incluse dans la spécification de contrôle S sous forme d'une arborescence d'états correspondante. Ces arborescences d'états sont traduites en prédicats par la logique de synthèse. Le détail de cette procédure est donné à la section 4.1 sur la définition de problèmes de contrôle dans [39]. C'est précisément cette forme de requis que l'on retrouve ici dans la catégorie des requis dits *standards*. Les requis utilisés ici sont directement transcrits sous forme de prédicats par commodité. Avec un modèle de STS sans structure verticale les prédicats sont d'ailleurs aussi expressifs et habituellement plus simple d'usage.

Les modèles de mémoire peuvent aussi inclure des transitions sur événements contrôlables. L'usage d'une transition sur événement contrôlable dans un modèle de mémoire constitue une altération directe de la fonction de transition du modèle du système global. C'est pourquoi on les considère comme des contraintes de contrôle *explicites* dans le modèle d'un sous-système. Pour expliciter ces requis (qui font déjà partie de la structure de transition du sous-système à travers leur inclusion dans celle d'un élément de mémoire) on utilise par exemple la forme `trsf <= Seq = 3` pour dénoter que l'événement contrôlable `trsf` requiert que l'élément de mémoire `Seq` soit à l'état 3.

Enfin, avec cette procédure de modélisation, une simple contrainte d'exclusion mutuelle entre deux familles d'états de dispositifs différents doit faire appel à un modèle mémoire. Or comme l'information existe déjà dans l'ensemble des modèles, pourquoi complexifier ce modèle par l'addition d'un modèle de mémoire reproduisant (par redondance) cette information ? Une discussion de cette forme de contrainte est donnée à la section 4.2.3.2. La spécification de la grue en figure 4.3 constitue un bon exemple de cette forme de contraintes qualifiées ici de contraintes *implicites* au modèle.

Finalement la section D.3 introduit la notion de *requis de contrôle élémentaire*. Simplement formulé, un requis de contrôle élémentaire correspond à une arborescence d'états. Dans un modèle de STS sans structure verticale, le prédicat correspondant à une arborescence d'états prend la forme d'une conjonction. Par exemple, en figure 4.3, chacun des prédicats de la figure est un requis de contrôle élémentaire. Pour des requis plus complexes tels que

```
/*(1)*/ SW = 0 ∧
        (Clamp ∈ {1,2,3} ∨ Mtr ∈ {1,2,3} ∨ Drv = 1)
```

extrait de la spécification de contrôle pour une perceuse en figure 4.17, il suffit de distribuer la conjonction sur l'ensemble des disjonctions. On a alors l'ensemble des trois requis de contrôle élémentaires suivants :

```
SW = 0 ∧ Clamp ∈ {1,2,3}
SW = 0 ∧ Mtr ∈ {1,2,3}
SW = 0 ∧ Drv = 1
```

en lieu et place de (1) dans la spécification de contrôle. Cette manoeuvre est justifiée par le fait qu'avec les STS, la spécification S est un ensemble d'arborescences d'états et que cet ensemble est transformé par la procédure de synthèse en une disjonction de prédicats représentant chacun des éléments de S (voir [39], section 4.1). Les requis de contrôle élémantaires constituent une bonne façon de *normaliser* les ensembles de requis de contrôle pour permettre une comparaison de leur *taille* comme mesure de leur complexité.

D.1 Composantes élémentaires

La spécification de composantes réutilisables élémentaires est décrite de façon détaillée à la section 4.2.2. Il peut être utile de s'y référer pour comprendre le contenu de cette section.

D.1.1 Jack111

Les vérins constituent une famille de dispositifs versatiles et admettent plusieurs variantes d'interfaces dénotées par les suffixes SS, cdcr, cd et cr. Le vérin *type*, la composante `Jack111`, détaille chacune de ces variantes. Pour tous les autres modèles de vérins donnés (y compris les vérins à commande bistable) il est présumé que ces variantes existent aussi dans la bibliothèque. Les interfaces sont exactement les mêmes que pour le vérin *type*, et les détails d'implémentation peuvent aisément être déduits des descriptions de ces variantes données pour le vérin *type*.

On trouve à la section 4.2.2 le modèle de composante réutilisable pour un vérin à commande monostable muni d'un ensemble complet de capteurs. Ce modèle appelé `Jack111` est l'un des plus simples et des plus complets pour cette catégorie de dispositifs ; on peut le

considérer comme un modèle de référence. La figure 4.9 donne la structure de sa projection, la figure 4.10 donne l'interface correspondante. La figure 4.8 donne des détails structurels complémentaires sur le sous-système associé à la composante réutilisable.

D.1.2 Jack111SS

La composante `Jack111SS` (SS pour *start/stop*), dont on trouve la définition d'interface en figure D.1, n'est qu'une simple variation de l'interface de la composante `Jack111`.

```
Basic Component Jack111SS<Y0, X0, X1> {
  SubSystem Jack111SS {
    State Vector = <Y0, X0, X1>
    f↑Y0 := ¬Y0 ∧ X0, f↓Y0 := Y0 ∧ X1
    Project↑Y0() := { return funcstart(); }
    Project↑X0() := { return funcstop(); }
    lal↑Y0() := { set(Y0); }
    lal↓Y0() := { rst(Y0); }
    cycle() := cycle() + {
      if ( ↑X0 ) then func↑X0();
      func↓Y0();
      func↓Y0();
    }
  }
}
```

D.1.3 Jack111cdcr

La composante `Jack111cdcr`, dont on trouve la définition d'interface en figure D.2, propose une autre variation de `Jack111`. Son interface permet d'interrompre le déploiement (événement *cd* pour *cancel deploy*) ou la rétractation (événement *cr* pour *cancel retract*) du vérin.

```
Basic Component Jack111cdcr<Y0, X0, X1> {

  SubSystem Jack111cdcr {
    State Vector = <Y0, X0, X1>
```

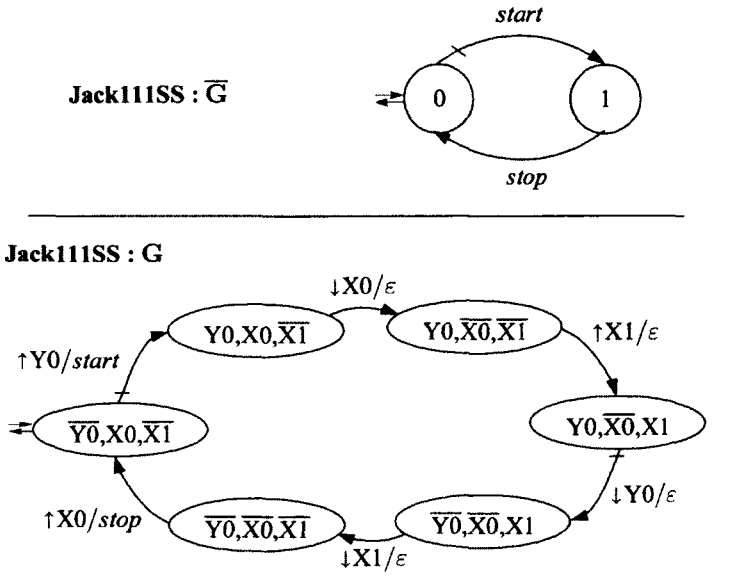


FIGURE D.1 – Composante *Jack111SS* (sous-système et projection)

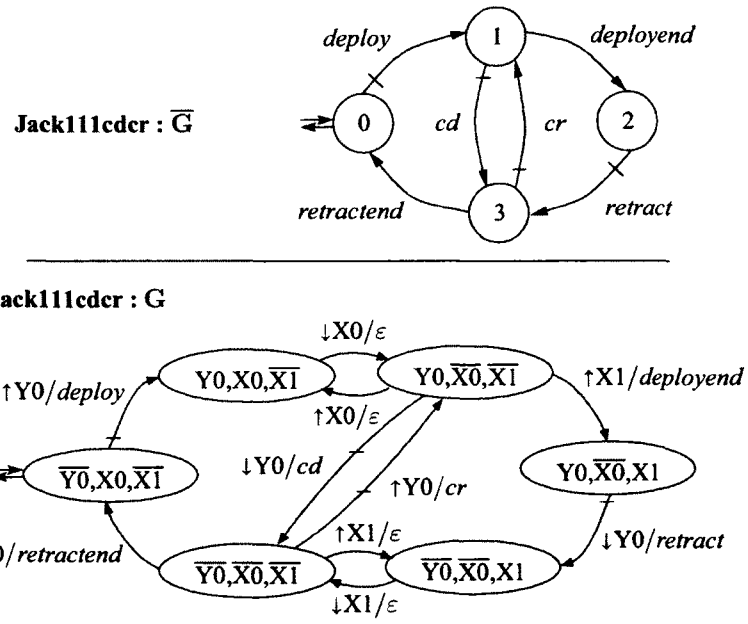


FIGURE D.2 – Composante *Jack111cdcr* (sous-système et projection)

```
    f↑Y0 := ¬Y0 ∧ ¬X1, f↓Y0 := Y0 ∧ ¬X0

    Project↑Y0() := {
      if ( X0 ) then return funcdeploy() else return funccr();
      return false; }
    Project↓Y0() := {
      if ( X1 ) then return funcretract() else return funccd();
      return false; }

    Project↑X1() := { return funcdeployend(); }
    Project↑X0() := { return funcretractend(); }

    lal↑Y0() := { set(Y0); }
    lal↓Y0() := { rst(Y0); }

    cycle() := cycle() + {
      if ( ↑X1 ) then func↑X1();
      if ( ↑X0 ) then func↑X0();
      func↑Y0();
      func↓Y0();
    }
  }
}
```

D.1.4 Jack111cd

La composante `Jack111cd`, dont on trouve la définition d'interface en figure D.3, propose une variante de l'interface de `Jack111cdcr` ou seul le déploiement du vérin peut être annulé.

```
Basic Component Jack111cd<Y0, X0, X1> {

  SubSystem Jack111cd {
    State Vector = <Y0, X0, X1>

    f↑Y0 := ¬Y0 ∧ X0, f↓Y0 := Y0 ∧ ¬X0

    Project↑Y0() := { return funcdeploy(); }
```

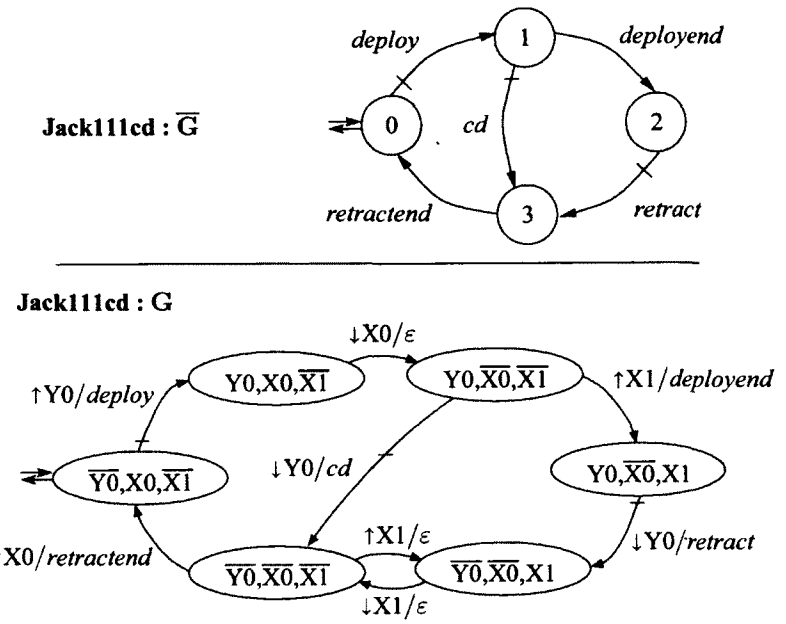


FIGURE D.3 – Composante *Jack111cd* (sous-système et projection)

```
    Project↓Y0() := {
      if ( X1 ) then return funcretract() else return funccd();
    }

    Project↑X1() := { return funcdeployend(); }
    Project↓X0() := { return funcretractend(); }

    lal↑Y0() := { set(Y0); }
    lal↓Y0() := { rst(Y0); }

    cycle() := cycle() + {
      if ( ↑X1 ) then func↑X1();
      if ( ↑X0 ) then func↑X0();
      func↑Y0();
      func↓Y0();
    }
  }
}
```

D.1.5 Jack111cr

La composante `Jack111cr`, dont on trouve la définition d'interface en figure D.4, propose une autre variante de l'interface de `Jack111cdcr` ou seule la rétractation du vérin peut être annulée.

```
Basic Component Jack111cr<Y0, X0, X1> {

  SubSystem Jack111cr {
    State Vector = <Y0, X0, X1>

    f↑Y0 := ¬Y0 ∧ ¬X1, f↓Y0 := Y0 ∧ X1

    Project↑Y0() := {
      if ( X0 ) then return func_deploy()
      else return func_cr(); }
    Project↓Y0() := { return func_retract(); }

    Project↑X1() := { return func_deployend(); }
    Project↑X0() := { return func_retractend(); }

    lal↑Y0() := { set(Y0); }
    lal↓Y0() := { rst(Y0); }

    cycle() := cycle() + {
      if ( ↑X1 ) then func↑X1();
      if ( ↑X0 ) then func↑X0();
      func↑Y0();
      func↓Y0();
    }
  }
}
```

D.1.6 Jack110

La composante `Jack110`, dont on trouve la définition d'interface en figure D.5, modélise un vérin à commande monostable ne disposant pas d'un capteur en fin de course. Le

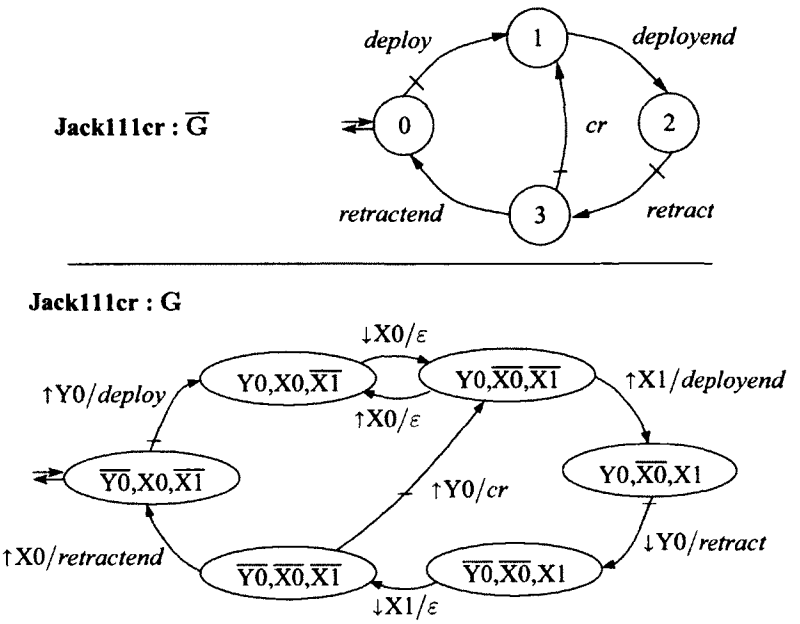


FIGURE D.4 – Composante *Jack111cr* (sous-système et projection)

capteur de fin de course du vérin est donc remplacé par une *minuterie mémorisée* de type ODT (voir section 4.2.1.3) dont l'état initial est *inactive*.

Il faut noter que la minuterie est *incorporée* au dispositif physique; il ne s'agit pas d'une composante réutilisable. Le modèle obtenu de façon *ad hoc* est en fait celui de la figure D.6. Ce dernier peut être ramené au modèle de la figure D.5. Cette hypothèse doit bien sûr faire l'objet d'une validation de la part du concepteur de la composante réutilisable. Cependant on peut noter que c'est l'élément de contrôle (l'ordinateur ou le PLC) qui est responsable de tout changement d'état d'un élément de mémoire. Par conséquent, à la figure D.6, l'événement ↓M1 se produit (au plus tard) au cycle de rétroaction suivant l'événement ↓Y0. Or, puisque ↑X0 est incontrôlable, il ne peut se produire qu'au prochain cycle de rétroaction (au plus tôt). Ainsi l'événement ↑X0 ne peut pas se produire avant ↓M1. Au pire les deux événements sont perçus comme simultanés.

```
Basic Component Jack110<Y0, X0, T1, dt : TIME> {

  /* Initial state = timer cleared */
  T1 : ODT[S := Y0 ∧ X0̄, M1̄, dt];
```

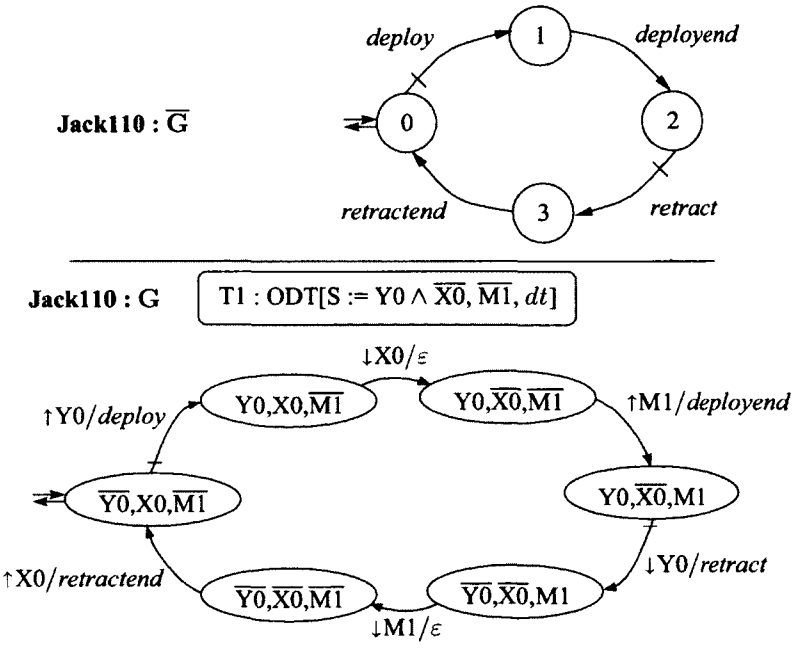


FIGURE D.5 – Composante *Jack110* (sous-système et projection)

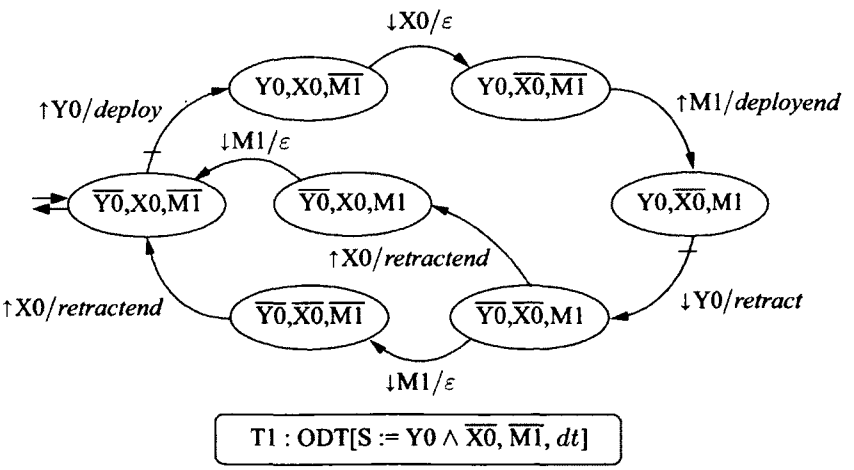


FIGURE D.6 – Composante *Jack110* (modèle exhaustif du sous-système)

```
SubSystem Jack110 {
  State Vector = <Y0, X0, M1>

  f_↑Y0 := ¬Y0 ∧ X0
  f_↓Y0 := Y0 ∧ M1

  Project_↑Y0() := { return func_deploy(); }
  Project_↓Y0() := { return func_retract(); }
  Project_↑M1() := { return func_deployend(); }
  Project_↑X0() := { return func_retractend(); }

  lal_↑Y0() := { set(Y0); }
  lal_↓Y0() := { rst(Y0); }
  lal_↑M1() := { set(M1); }
  lal_↓M1() := { rst(M1); }

  cycle() := cycle() + {
    if (( ¬Y0 ∨ X0 ) ∧ M1 ) then func_↓M1();
    if ( T1 ∧ Y0 ∧ ¬X0 ∧ ¬M1) then func_↑M1();
    eval_f(T1, Y0 ∧ ¬X0 ∧ ¬M1);
    if ( ↑X0 ) then func_↑X0();
    func_↑Y0();
    func_↓Y0();
  }
 }
}
```

D.1.7 Jack101

La composante réutilisable `Jack101`, dont on retrouve la définition d'interface en figure D.7, modélise un vérin à commande monostable ne disposant pas de capteur de début de course. Ce capteur est donc remplacé par une minuterie mémorisée de type ODT dont l'état initial est *échue*.

```
Basic Component Jack101<Y0, T0, X1, dt : TIME> {

  /* Initial state = timer expired */
```

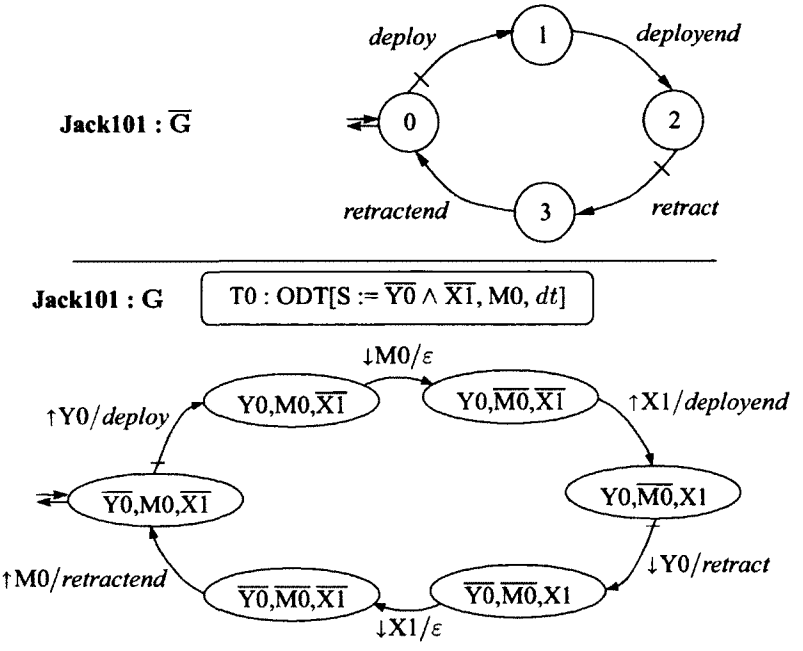


FIGURE D.7 – Composante *Jack101* (sous-système et projection)

```
T0 : ODT[S := Y0 ∧ X1, M0, dt];

SubSystem Jack101 {
  State Vector = <Y0, M0, X1>

  f↑Y0 := Y0 ∧ M0
  f↓Y0 := Y0 ∧ X1

  Project↑Y0() := { return funcdeploy(); }
  Project↓Y0() := { return funcretract(); }
  Project↓X1() := { return funcdeployend(); }
  Project↑M0() := { return funcretractend(); }

  lal↑Y0() := { set(Y0); }
  lal↓Y0() := { rst(Y0); }
  lal↑M0() := { set(M0); }
  lal↓M0() := { rst(M0); }
```

```
cycle() := cycle() + {
  if ((Y0 ∨ X1) ∧ M0 ) then func↓M0();
  if ( T0 ∧ Ȳ0 ∧ X̄1 ∧ M̄0 ) then func↑M0();
  evalT(T0, Ȳ0 ∧ X̄1 ∧ M̄0);
  if ( ↑X1 ) then func↑X1();
  func↑Y0();
  func↓Y0();
} } }
```

D.1.8 Jack100

La composante `Jack100`, dont on retrouve la définition d'interface en figure D.8, modélise un vérin à commande monostable ne disposant d'aucun capteur. Les capteurs sont remplacées par deux minuteries mémorisées de type ODT dont l'une est à l'état initial *échue* et l'autre à l'état initial *inactive*. Les délais de réaction au déploiement et à la rétractation du vérin peuvent être asymétriques et même éventuellement nuls, pourvu que la machine cible supporte correctement un délai nul sur une *minuterie brute*.

```
Basic Component Jack100<Y0, T0, dt0  : TIME,
                         T1, dt1 : TIME> {

  /* Initial state = expired */
  T0 : ODT[S := Ȳ0 ∧ M̄1, M0, dt0];
  /* Initial state = cleared */
  T1 : ODT[S := Y0 ∧ M̄0, M̄1, dt1];

  SubSystem Jack101 {
    State Vector = <Y0, M0, M1>

    f↑Y0 := Ȳ0 ∧ M0
    f↓Y0 := Y0 ∧ M1

    Project↑Y0() := { return func_deploy(); }
    Project↓Y0() := { return func_retract(); }
    Project↑M1() := { return func_deployend(); }
    Project↑M0() := { return func_retractend(); }
```

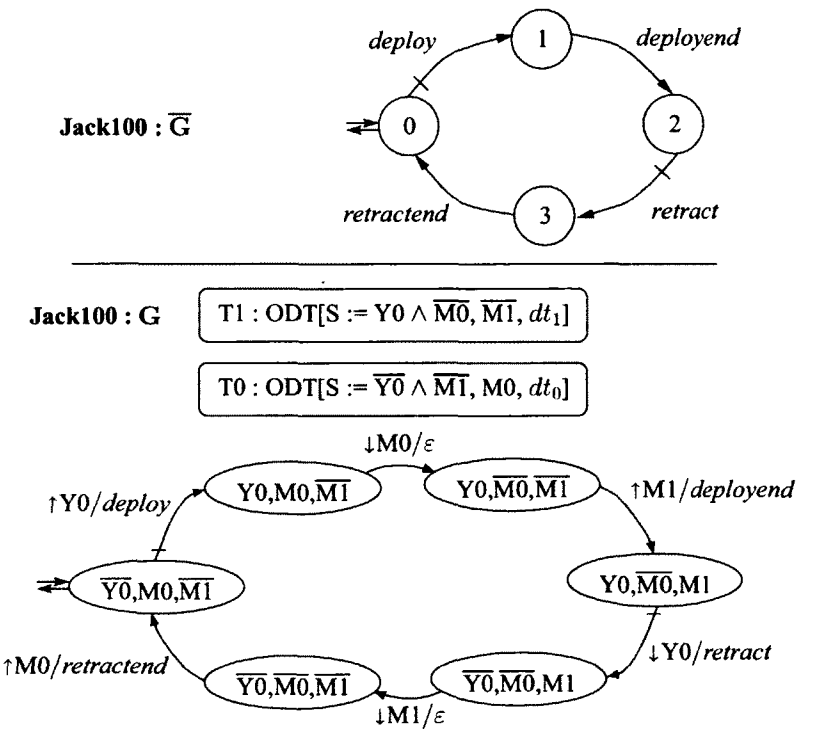


FIGURE D.8 – Composante *Jack100* (sous-système et projection)

```
    lal↑Y0() := { set(Y0); }
    lal↓Y0() := { rst(Y0); }
    lal↑M0() := { set(M0); }
    lal↓M0() := { rst(M0); }
    lal↑M1() := { set(M1); }
    lal↓M1() := { rst(M1); }

    cycle() := cycle() + {
      if ((Y0 ∨ M1) ∧ M0 ) then func↓M0();
      if ((¬Y0 ∨ M0) ∧ M1 ) then func↓M1();

      if ( T0 ∧ ¬Y0 ∧ ¬M1 ∧ ¬M0 ) then func↑M0();
      evalT(T0, ¬Y0 ∧ ¬M1 ∧ ¬M0);

      if ( T1 ∧ Y0 ∧ ¬M0 ∧ ¬M1 ) then func↑M1();
      evalT(T1, Y0 ∧ ¬M0 ∧ ¬M1);

      func↑Y0();
      func↓Y0();
    }
  }
}
```

D.1.9 Jack211a

La composante `Jack211a`, dont on retrouve la définition d'interface en figure D.9, modélise un vérin à commande bistable *verrouillée* (équivalent anglais pour *latched*).

Il existe deux types de vérins bistables :

1. le vérin dit bistable (sans qualification) dont la commande est alors présumée *non verrouillée*) ;
2. le vérin bistable à commande *verrouillée*.

La position d'un vérin est dite *stable* si une coupure de l'alimentation à la commande n'en change pas la position. Un vérin bistable possède deux positions dites stables. Les vérins bistables dont la commande est *verrouillée* complètent toujours la dernière commande donnée, même si l'alimentation à la commande est coupée en cours de fonctionnement.

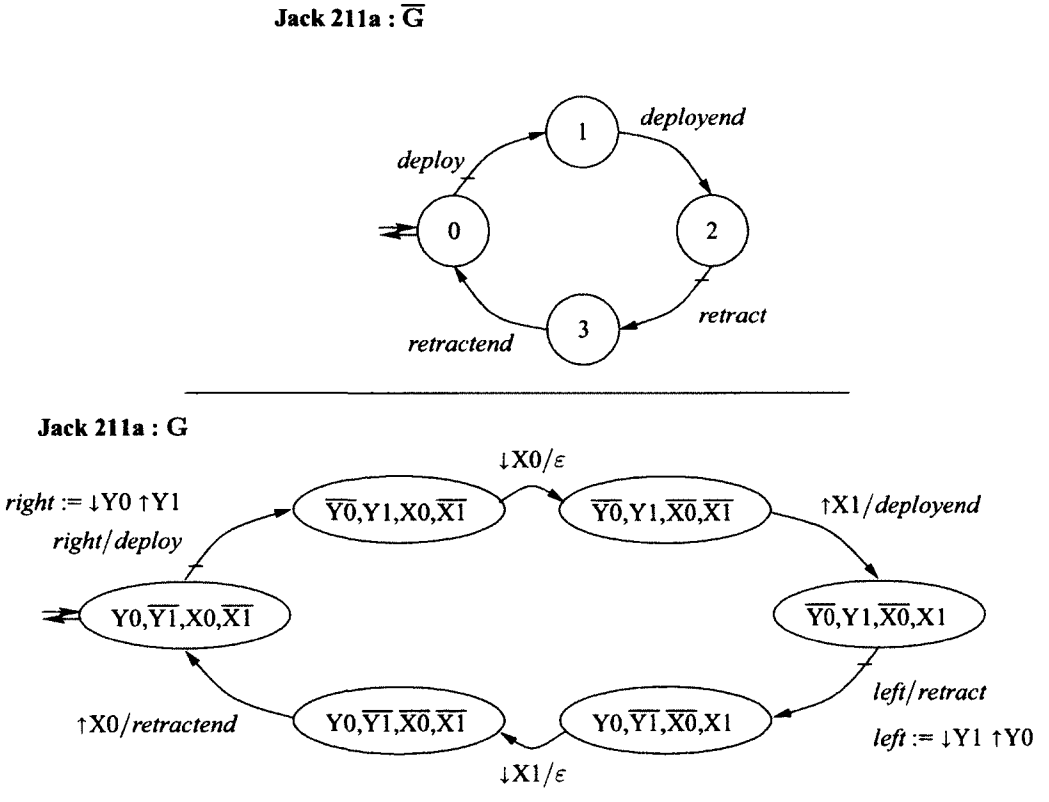


FIGURE D.9 – Composante *Jack211a* (sous-système et projection)

Ils se stabilisent ainsi toujours dans une position connue. Il est toutefois possible d'inverser (annuler) la dernière commande avant que le vérin n'ait eu le temps de se stabiliser. Les vérins bistables (à commande *non verrouillée*) s'immobilisent immédiatement sur une interruption de l'alimentation à la commande. Ils peuvent donc occuper une position intermédiaire *inconnue* à la restauration de l'alimentation. Puisque cette catégorisation des vérins bistables concerne leur comportement en régime de panne, que cette étude ignore par hypothèse, tous les vérins de ce type sont modélisés ici comme des vérins à commande bistable verrouillée.

```
Basic Component Jack211a<Y0, Y1, X0, X1> {

  SubSystem Jack211a {
    State Vector = <Y0, Y1, X0, X1>

    f_right := Y0 ∧ X0
    f_left := Y1 ∧ X1

    Project_right() := { return func_deploy(); }
    Project_↑X1() := { return func_deployend(); }
    Project_left() := { return func_retract(); }
    Project_↑X0() := { return func_retractend(); }

    lal_right() := { rst(Y0); set(Y1); }
    lal_left() := { rst(Y1); set(Y0); }

    cycle() := cycle() + {
      if ( ↑X1 ) then func_↑X1();
      if ( ↑X0 ) then func_↑X0();
      func_right();
      func_left();
    }
  } }
```

D.1.10 Jack211b

Un vérin à commande bistable ayant deux positions stables, chacune de ces positions peut être spécifiée comme état initial du modèle. La composante `Jack211b`, dont on re-

trouve la définition d'interface en figure D.10, propose simplement une version alternative de la définition de l'interface de la composante `Jack211a`.

Le code d'intégration pour la composante `Jack211b` demeure le même que pour la composante `Jack211a` puisque la seule chose qui soit différente dans le modèle est l'état initial.

D.1.11 ADRelay

La composante `ADRelay` (pour *Asymmetric Delay Relay*), dont on retrouve la définition d'interface en figure D.11, propose un modèle général pour des dispositifs activés par un relais électrique qui ne sont pas munis de capteurs. Ces dispositifs sont mis *en service* ou *hors service* de façon commandée et ont généralement des délais de fonctionnement (au démarrage et à l'arrêt) éventuellement asymétriques. Par exemple la composante `Jack100` peut être modélisée par instanciation de `ADRelay`. Aussi cette composante admet des variantes d'interfaces similaires à celles données pour les vérins avec les suffixes suivant :

- SS (*start/stop*) ;
- cc (*cancel close*) ;
- co (*cancel open*) ;
- et ccco (annulation aux deux commandes).

Un moteur électrique ayant un délai de latence au démarrage pour atteindre sa vitesse nominale constitue un meilleur exemple d'une instance de cette composante. Finalement le modèle est viable même lorsque l'un, l'autre ou les deux délais sont nuls.

```
Basic Component ADRelay<Y0, T0, dt0  : TIME,
                         T1, dt1 : TIME> {

  /* Initial state = expired */
  T0 : ODT[S := Y0 ∧ M1, M0, dt0];
  /* Initial state = cleared */
  T1 : ODT[S := Y0 ∧ M0, M1, dt1];

  SubSystem ADRelay {
    State Vector = <Y0, M0, M1>
```

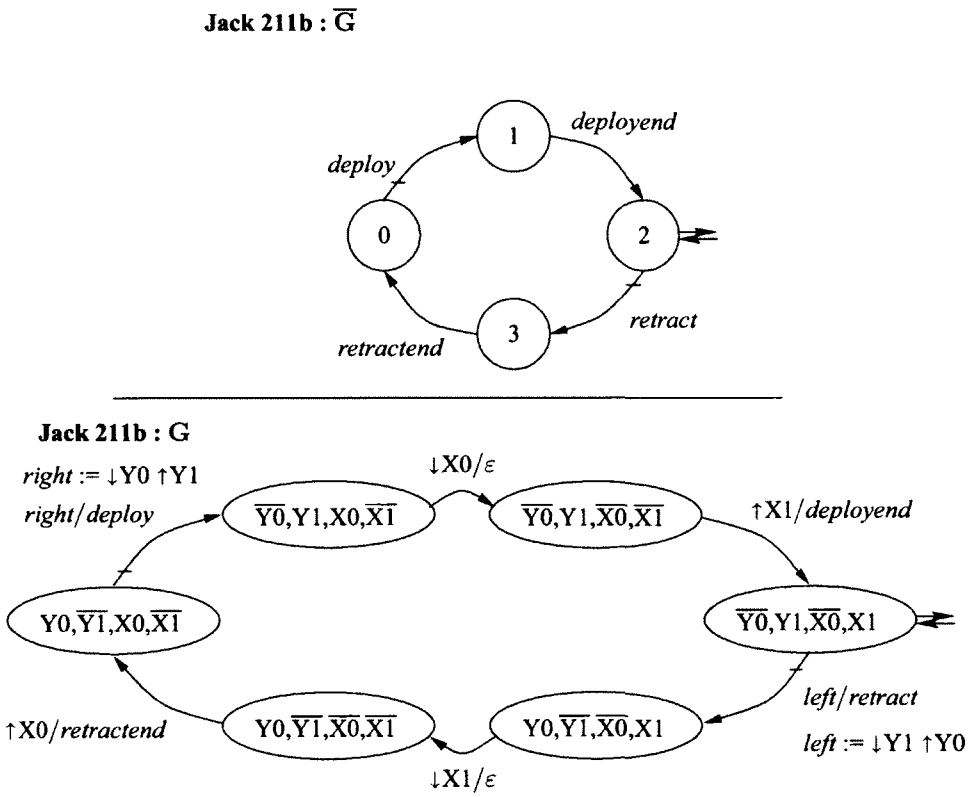


FIGURE D.10 – Composante *Jack211b* (sous-système et projection)

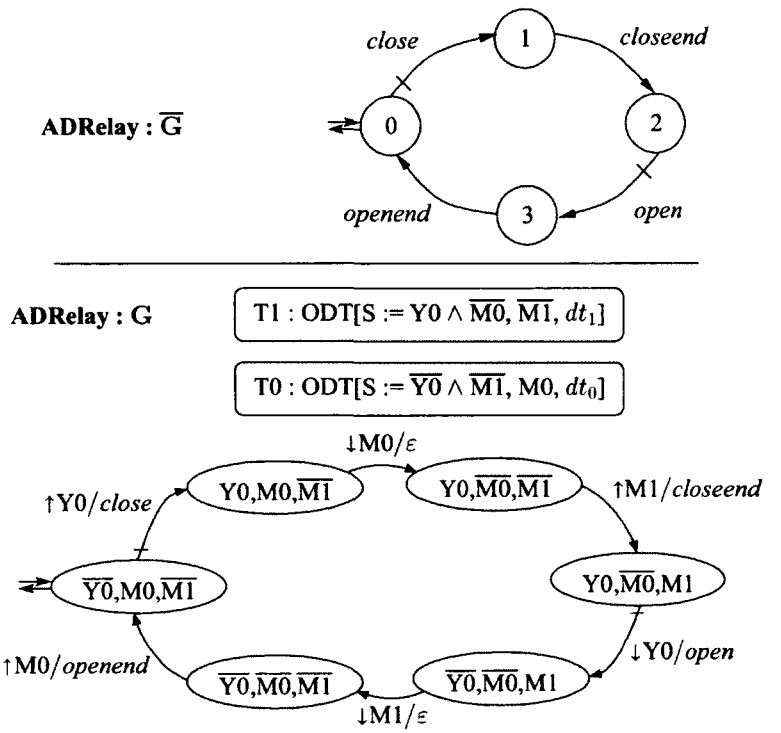


FIGURE D.11 – Composante *ADRelay* (sous-système et projection)

```
    f_↑Y0 := ¬Y0 ∧ M0
    f_↓Y0 := Y0 ∧ M1

    Project_↑Y0() := { return func_close(); }
    Project_↓Y0() := { return func_open(); }
    Project_↑M1() := { return func_closeend(); }
    Project_↑M0() := { return func_openend(); }

    lal_↑Y0() := { set(Y0); }
    lal_↓Y0() := { rst(Y0); }
    lal_↑M0() := { set(M0); }
    lal_↓M0() := { rst(M0); }
    lal_↑M1() := { set(M1); }
    lal_↓M1() := { rst(M1); }

    cycle() := cycle() + {
      if ((Y0 ∨ M1) ∧ M0 ) then func_↓M0();
      if ((¬Y0 ∨ M0) ∧ M1 ) then func_↓M1();

      if ( T0 ∧ ¬Y0 ∧ ¬M1 ∧ ¬M0 ) then func_↑M0();
      eval_I'(T0, ¬Y0 ∧ ¬M1 ∧ ¬M0);

      if ( T1 ∧ Y0 ∧ ¬M0 ∧ ¬M1 ) then func_↑M1();
      eval_I'(T1, Y0 ∧ ¬M0 ∧ ¬M1);

      func_↑Y0();
      func_↓Y0();
    }
  }
}
```

D.1.12 SDRelay

La composante `SDRelay` (pour *Symmetric Delay Relay*), dont on retrouve la définition d'interface en figure D.12, reprend `ADRelay` mais en utilisant une minuterie à double action pour procurer des délais symétriques. Bien que cette composante admette une interface de type *démarrage-arrêt* (suffixe SS), l'usage d'une seule minuterie interdit les autres

formes d'interfaces (cc, co et ccco). Cette composante admet aussi l'usage d'un délai nul.

```
Basic Component SDRelay<Y0, T0, dt : TIME> {

  /* Initial state = cleared */
  T0 : DDT[S := ((¬Y0 ∧ M0) ∨ (Y0 ∧ ¬M0)), ¬M0, dt];

  SubSystem SDRelay {
    State Vector = <Y0, M0>

    f↑Y0 := ¬Y0 ∧ ¬M0
    f↓Y0 := Y0 ∧ M0

    Project↑Y0() := { return funcclose(); }
    Project↓Y0() := { return funcopen(); }
    Project↑M0() := { return funcCloseend(); }
    Project↓M0() := { return funcOpenend(); }

    lal↑Y0() := { set(Y0); }
    lal↓Y0() := { rst(Y0); }
    lal↑M0() := { set(M0); }
    lal↓M0() := { rst(M0); }

    cycle() := cycle() + {
      if ( T0 ∧ Y0 ∧ ¬M0 ) then func↑M0();
      if ( T0 ∧ ¬Y0 ∧ M0 ) then func↓M0();
      evalT(T0, ((¬Y0 ∧ M0) ∨ (Y0 ∧ ¬M0)));

      func↑Y0();
      func↓Y0();
    }
  } }
```

D.1.13 NDRelay

La composante NDRelay (pour *No Delay Relay*), dont on retrouve la définition d'interface en figure D.13, n'est qu'une version sans temporisation de SDRelay n'utilisant aucune minuterie. Elle admet aussi une interface de type *démarrage-arrêt* (suffixe SS).

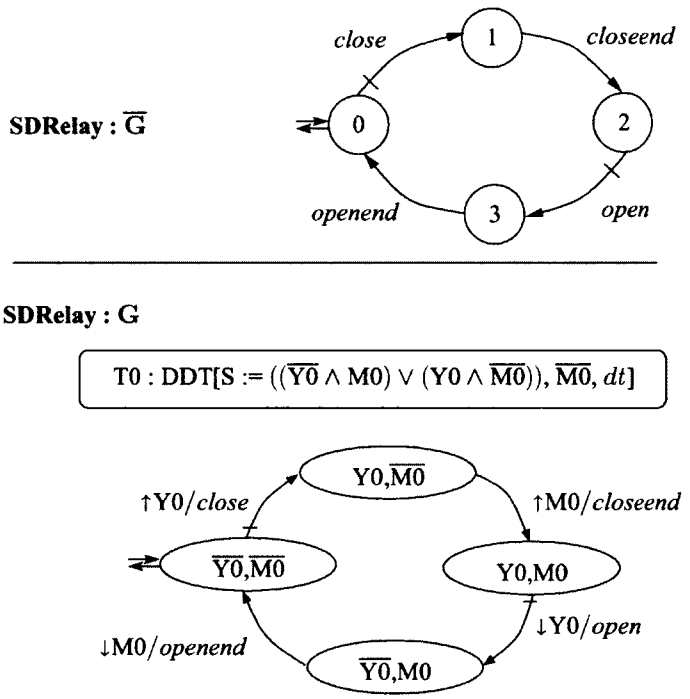


FIGURE D.12 – Composante *SDRelay* (sous-système et projection)

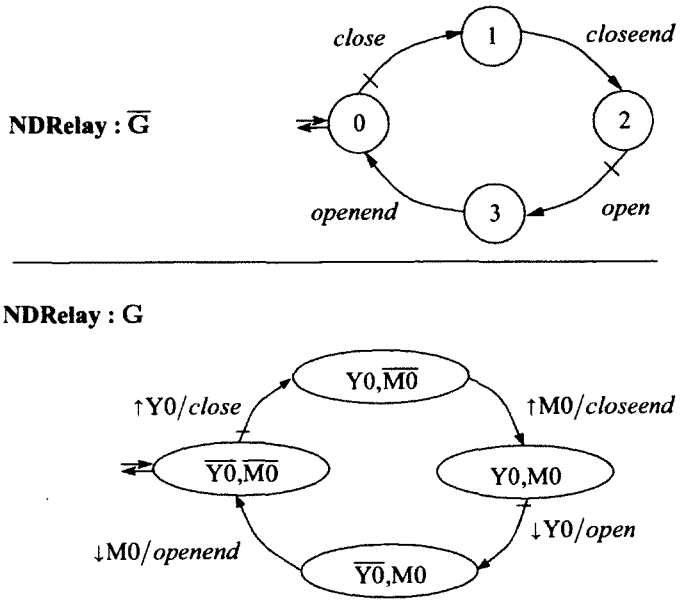


FIGURE D.13 – Composante *NDRelay* (sous-système et projection)

```
Basic Component NDRelay<Y0> {

  SubSystem NDRelay {
    State Vector = <Y0, M0>

    f↑Y0 := ¬Y0 ∧ ¬M0
    f↓Y0 := Y0 ∧ M0

    Project↑Y0() := { return func_close(); }
    Project↓Y0() := { return func_open(); }
    Project↑M0() := { return func_closeend(); }
    Project↓M0() := { return func_openend(); }

    lal↑Y0() := { set(Y0); }
    lal↓Y0() := { rst(Y0); }
    lal↑M0() := { set(M0); }
    lal↓M0() := { rst(M0); }

    cycle() := cycle() + {
```

```
      if ( Y0 ∧ M0 ) then func↑M0();
      if ( Y0 ∧ M0 ) then func↓M0();
      func↑Y0();
      func↓Y0();
    }
  }
}
```

D.1.14 ODTimer

La composante `ODTimer` (pour *On Delay Timer*), dont on retrouve la définition d'interface en figure D.14, propose d'encapsuler le comportement d'une minuterie mémorisée de type ODT sous forme d'une composante réutilisable. L'avantage d'une minuterie sous forme de composante est de permettre l'usage de la synthèse SCT dans des systèmes conçus par assemblage de composantes réutilisables (par opposition à la conception *ad hoc*). Par défaut ce type de minuterie est *normalement inactive* (son état initial est *inactive* c'est-à-dire 0), et on peut annuler le minutage (transition sur *rst* dans l'interface). On peut aisément donner les variantes d'interfaces suivantes pour cette minuterie :

1. ODTimerNC (pour *Non Cancellable*), normalement inactive *sans* annulation du minutage (pas de commande *rst* dans l'interface) ;
2. ODTimerNE (pour *Normally Expired*), normalement expirée *avec* annulation du minutage (état initial *expirée*, c'est-à-dire 2) ;
3. ODTimerNCNE, normalement expirée *sans* annulation du minutage (état initial *expirée* et pas de commande *rst* dans l'interface).

On présume que la bibliothèque de composantes réutilisables contient toutes ces variantes.

```
Basic Component ODTimer<T0, dt : TIME> {

  /* Initial state = cleared */
  T0 : ODT[S := M0, M1, dt];

  SubSystem ODTimer {
    State Vector = <M0, M1>
```

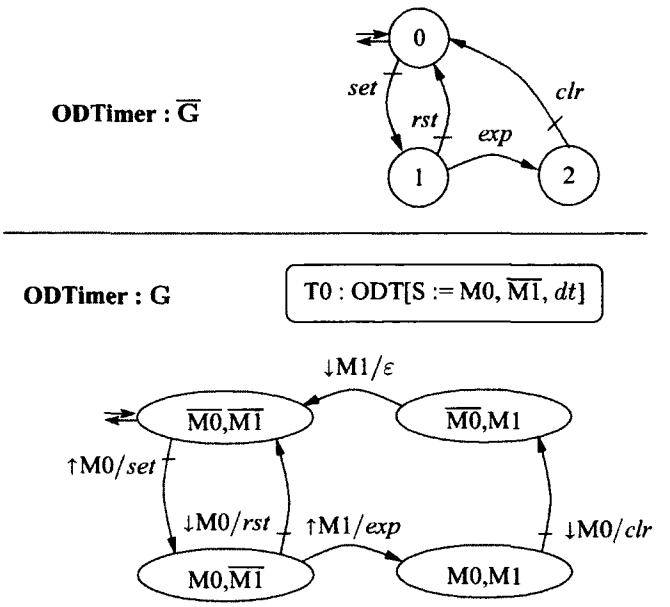


FIGURE D.14 – Composante *ODTimer* (sous-système et projection)

```
f↑M0 := M0 ∧ M1
f↓M0 := M0

Project↑M0() := { return func_set(); }
Project↓M0() := {
  if ( M1 ) then return func_clr();
  else return func_rst(); }
Project↑M1() := { return func_exp(); }

lal↑M0() := { set(M0); }
lal↓M0() := { rst(M0); }
lal↑M1() := { set(M1); }
lal↓M1() := { rst(M1); }

cycle() := cycle() + {
  if ( M0 ∧ M1 ) then func↓M1();
  if ( T0 ∧ M0 ∧ M1 ) then func↑M1();
  eval_T(T0, M0 ∧ M1);
  func↑M0();
```

```
        func↓M0();
      }
    }
}
```

D.1.15 Injector

Certains dispositifs de l'usine *MPS* se *composent* d'éléments matériels qui ont été modélisés précédemment mais dont l'assemblage résulte en un ensemble d'interactions trop *fines* pour être modélisés par assemblage et synthèse de composantes réutilisables. De tels dispositifs doivent ainsi être modélisés de façon *ad hoc*. C'est le cas de la composante élémentaire `Injector` dont on retrouve la définition d'interface à la figure D.15.

L'injecteur est constitué d'un vérin à commande monostable couplé physiquement à un barillet doté d'un capteur situé à la base d'une colonne cylindrique servant de magasin. Les pièces sont introduites par gravité dans le barillet où leur présence est détectée par un capteur optique (X0). Lorsque le capteur est à l'état $\overline{X0}$ c'est qu'une pièce obstrue le faisceau lumineux. Les pièces sont *injectées* sur un dock de chargement et y sont maintenues par le vérin jusqu'à ce que le vérin soit réarmé. Pendant l'injection il y a possibilité d'une interaction nuisible à la plateforme de chargement. Cette opération doit donc être visible dans l'état du dispositif. Le requis principal de l'injecteur est de n'injecter une pièce que s'il y a effectivement une pièce présente dans le barillet.

```
Basic Component Injector<X0, Y0, X1, X2> {

    SubSystem Injector {
      State Vector = <X0, Y0, X1, X2>

      f↑Y0 := ¬X0 ∧ ¬Y0 ∧ X1
      f↓Y0 := Y0 ∧ X2

      Project↑Y0() := { return func_inject(); }
      Project↑X2() := { return func_injectend(); }
      Project↓Y0() := { return func_rearm(); }
      Project↓X0() := { return func_wpa(); }
```

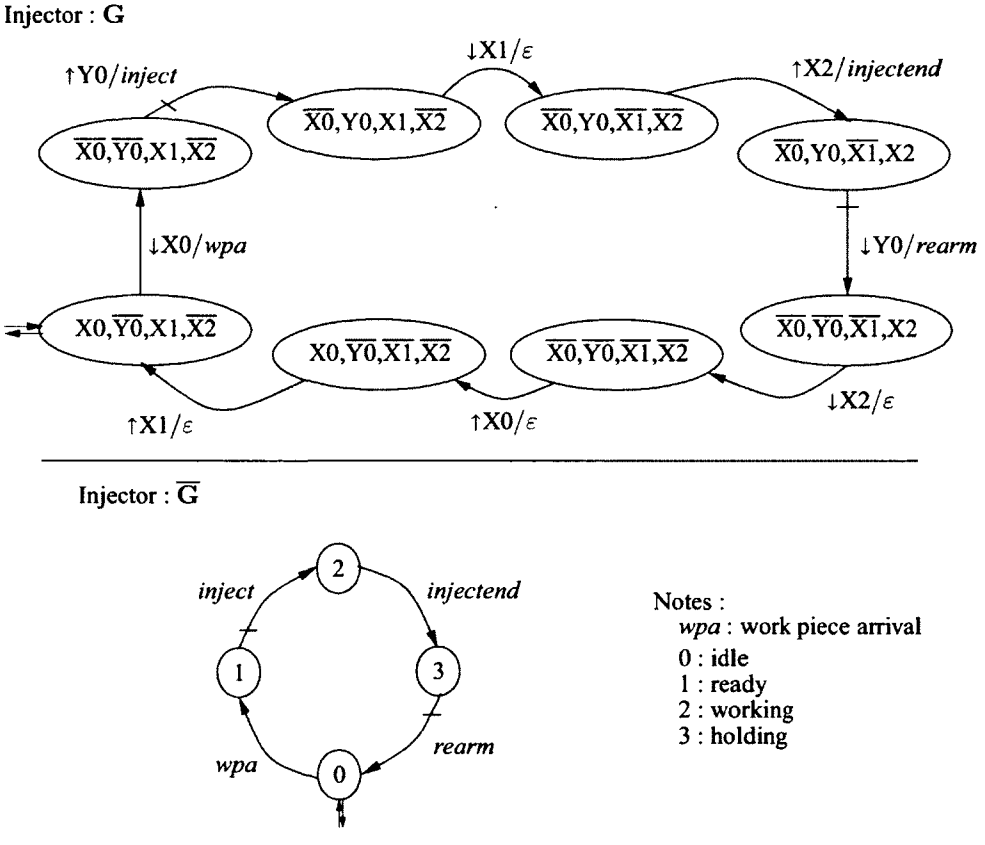


FIGURE D.15 – Composante *Injector* (sous-système et projection)

```
      lal↑Y0() := { set(Y0); }
      lal↓Y0() := { rst(Y0); }

      cycle() := cycle() + {
        if ( ↑X2 ) then func↑X2();
        if ( ↓X0 ) then func↓X0();
        func↑Y0();
        func↓Y0();
      }
    }
  }
```

D.1.16 Stepper

La composante `Stepper`, dont on retrouve la définition d'interface en figure D.16, modélise un mécanisme d'entraînement typique des convoyeurs. Chaque démarrage entraîne la progression du convoyeur d'*une* position dans un sens déterminé. Dû à la sévérité des contraintes temporelles impliquées entre le capteur d'alignement et le mécanisme d'entraînement, l'arrêt du dispositif doit être automatique (décidé et effectué localement) et soustrait à l'effet d'un contrôle externe.

```
Basic Component Stepper<Y0, X0> {

  SubSystem Stepper {
    State Vector = <Y0, X0>

    f↑Y0 := ¬Y0

    Project↑Y0() := { return func_start(); }
    Project_end() := { return func_stop(); }

    lal↑Y0() := { set(Y0); }
    lal_end() := { rst(Y0); }

    cycle() := cycle() + {
      if ( ↑X0 ) then func_end();
      func↑Y0();
```

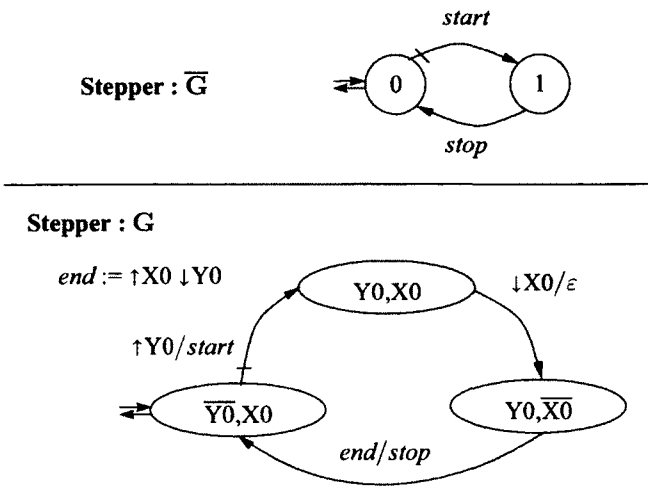


FIGURE D.16 – Composante *Stepper* (sous-système et projection)

```
    }
  }
}
```

D.1.17 SuctionCup et SuctionCupCC

La composante `SuctionCup`, dont on retrouve la définition d'interface en figure D.17, modélise habituellement la pince d'une grue (un dispositif de préhension). Il s'agit d'une pompe à vide à commande bistable munie d'un capteur de différentiel de pression.

```
Basic Component SuctionCup<Y0, Y1, X0> {

  SubSystem SuctionCup {
    State Vector = <Y0, Y1, X0>

    f_on := Y0 ∧ ¬X0, f_off := Y1 ∧ X0

    Project_on() := { return func_close(); }
    Project_↑X0() := { return func_closeend(); }
    Project_off() := { return func_open(); }
    Project_↓X0() := { return func_openend(); }
```

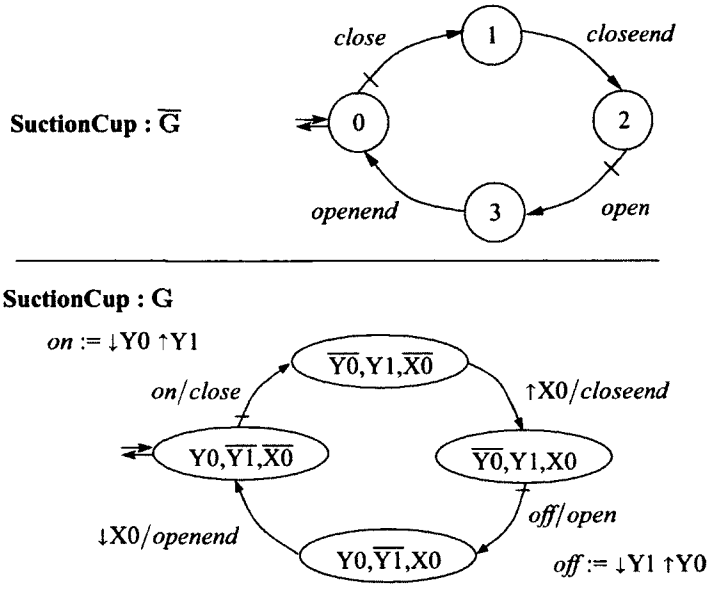


FIGURE D.17 – Composante *SuctionCup* (sous-système et projection)

```
    lal_on() := { rst(Y0); set(Y1); }
    lal_off() := { rst(Y1); set(Y0); }

    cycle() := cycle() + {
      if ( ↑X0 ) then func_↑X0();
      if ( ↓X0 ) then func_↓X0();
      func_on();
      func_off();
    }
  }
}
```

La figure D.18 suggère une version alternative dont la commande de fermeture peut être annulée.

Le code d'intégration de `SuctionCupCC` ne demande que des altérations mineures à celui donné pour `SuctionCup`. Il s'agit de redéfinir le prédicat de contrôle et la fonction $Project_{off}()$ de la façon suivante.

```
  ...
  f_off := Y1
```

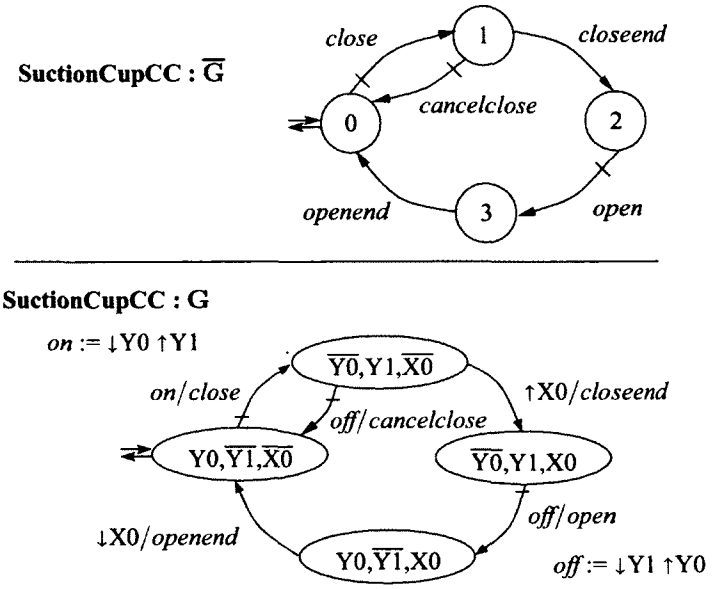


FIGURE D.18 – Composante *SuctionCupCC* (sous-système et projection)

```
...
Project_off() := {
  if ( X0 ) then return func_open();
  else return func_cancelclose(); }
...
```

Le reste du code demeure inchangé.

D.2 Composantes mixtes

La spécification de composantes réutilisables mixtes est décrite de façon détaillée à la section 4.2.3. Il peut être utile de s'y référer pour comprendre le contenu de cette section.

D.2.1 LRCrane et BLRCrane

On trouve, à la section 4.2.3, le modèle de composante réutilisable pour une grue à sens unique transportant des pièces à traiter d'une position d'entrée à une position de sortie. La grue est composée d'une flèche, pour les mouvements latéraux, et d'une pince. La

flèche (en anglais *Boom*) est instanciée à partir de la composante élémentaire `Jack211b`. La pince (en anglais *Hook*, une ventouse pneumatique) est instanciée à partir de la composante élémentaire `SuctionCup`. Le modèle de cette grue est appelé `LRCrane` (pour *Left to Right Crane*). La figure 4.12 donne un modèle de sa projection G ainsi qu'un modèle de son interface $\overline{\mathrm{G}}$ avec (dans la partie inférieure) un diagramme d'assemblage. La figure 4.13 donne des détails structurels complémentaires sur le sous-système associé à la composante réutilisable : structure du vecteur d'état, énoncés d'instanciation, spécification de contrôle ainsi que l'ensemble des prédicats de contrôle résultant de la synthèse à partir de la spécification de contrôle.

Avec la même structure de sous-système (celle de la figure 4.13) on peut aussi offrir l'interface définie par le modèle de la projection de la figure D.19 pour la composante `BLRCrane`. Ce type de projection (où tous les événements sont systématiquement projetés) est trivialement un *observateur* puisqu'il y a clairement une relation de bisimulation entre $\overline{\mathrm{G}}$ (le modèle de la projection résultante) et G_{ts} (la structure de transition de la machine de Mealy G). On peut d'ailleurs faire cette manoeuvre pour n'importe quel modèle afin de bénéficier d'une encapsulation des requis de contrôle par la dynamique abstraite.

D.2.2 Drill

On trouve, à la section 4.2.3, le modèle de composante réutilisable pour une perceuse. La perceuse est composée de trois dispositifs :

- un étau (élément *Clamp*) pour stabiliser les pièces à percer (instance de la composante élémentaire `Jack111`) ;
- un moteur électrique (élément *Mtr*) pour actionner la mèche de la perceuse (instance de la composante élémentaire `SDRelay`) ;
- et un patin (élément *Drv*) sur lequel est fixé le dispositif de perçage pour les mouvements verticaux (instance de la composante élémentaire `Jack211aSS`).

Deux éléments de mémoire sont utilisés pour coordonner l'activité des dispositifs physiques formant la perceuse :

1. SW (pour *SWitch*) indiquant l'état du dispositif (*en fonction* ou *hors fonction*) ;
2. PHS (pour *PHaseS*) indiquant le progrès du perçage.

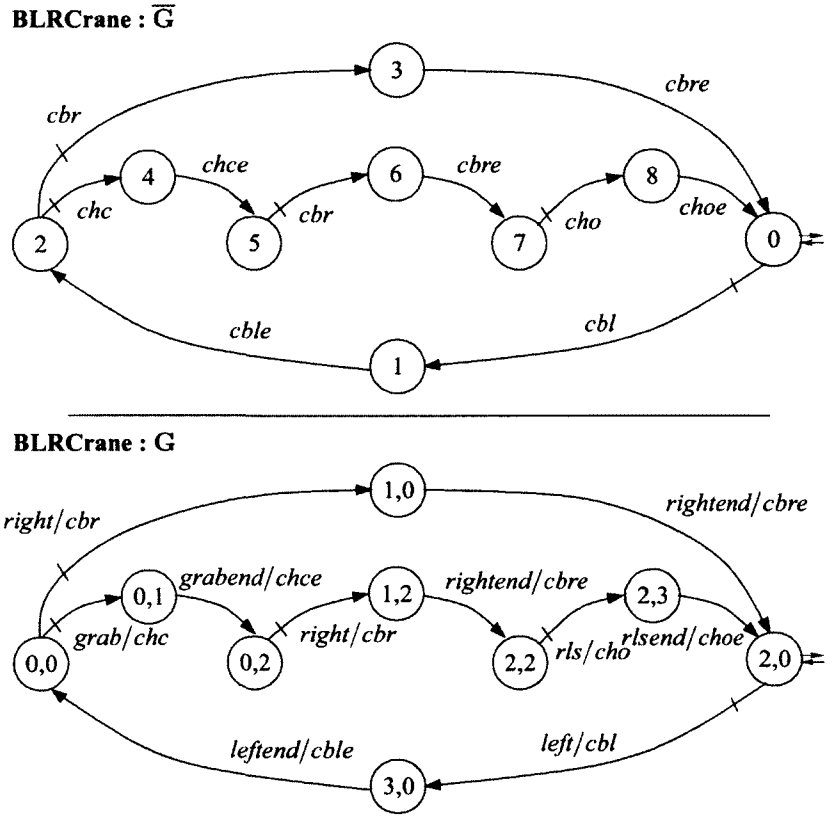


FIGURE D.19 – Composante *BLRCrane* (sous-système et projection)

L'ensemble est abstrait en une composante réutilisable appelée *Drill* avec *commande au démarrage* et *indication à arrêt*.

La figure 4.14 donne un diagramme d'assemblage du sous-système. La figure 4.15 donne un modèle de sa projection G ainsi qu'un modèle de son interface $\overline{G}$. La figure 4.16 donne les détails structurels du sous-système associé à la composante : structure du vecteur d'état, déclaration des mémoires, énoncés d'instanciation et événements *synthétiques* générés par le sous-système. La figure 4.17 donne la spécification de contrôle ainsi que loi de contrôle résultant de la synthèse pour le sous-système. L'ensemble des requis peut être décrite comme suit.

(1) Aucun dispositif ne doit être en fonction lorsque le sous-système est hors fonction.

(2) Pendant la phase de perçage, l'étau ne doit pas amorcer une ouverture ni le moteur un arrêt.

(3) Une fois le perçage terminé, l'étau ne doit pas amorcer de serrage ni le moteur initier un démarrage.

(4) Une fois le perçage terminé, le patin ne doit pas réamorcer de descente.

(5) Le patin ne doit pas entrer en fonction à moins que l'étau ne soit complètement fermé et le moteur démarré et à vitesse nominale.

Les requis (1) à (4) inclusivement sont standards et le requis (5) est implicite.

D.2.3 MicroMeter

On trouve à la figure D.20 la composante réutilisable *MicroMeter* dont un diagramme d'assemblage est donné en figure D.21.

Le micromètre est un dispositif composé d'une sonde montée sur un vérin que l'on peut abaisser sur un élément d'usinage (lorsqu'il est correctement disposé) pour en mesurer l'épaisseur. La sonde est connectée à un convertisseur analogique à digital muni d'un registre d'entrée donné en paramètre. L'épaisseur des éléments d'usinage doit se situer dans une plage de mesure dont les bornes sont données en paramètres.

La sonde doit d'abord être engagée complètement pour que la lecture soit valide. Une minuterie est utilisée pour éliminer les transitoires à l'arrivée de la sonde en position. La

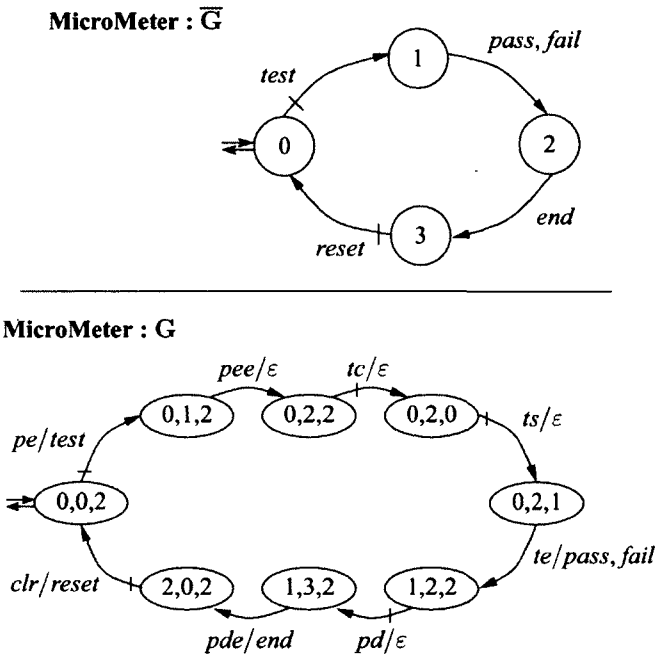


FIGURE D.20 – Composante *MicroMeter* (sous-système et projection)

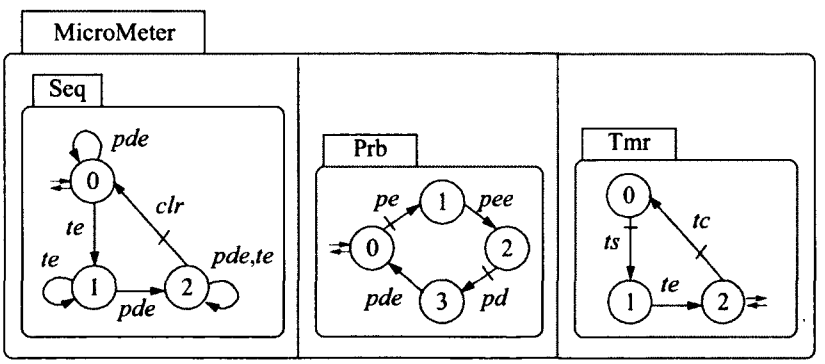


FIGURE D.21 – Composante *MicroMeter* (diagramme d'assemblage)

lecture est prise à l'expiration de la minuterie et un diagnostic de test de tolérance est émis sous forme d'événements : *pass* lorsque l'épaisseur se situe à l'intérieur de la tolérance acceptable, *fail* sinon. La sonde doit être maintenue en position de lecture jusqu'à l'émission du diagnostic de test, puis désengagée. Une fois un test effectué et la sonde désengagée, le micromètre ne peut être réutilisé qu'après avoir été réarmé. Ce qui se traduit par l'ensemble des requis de contrôle suivants.

(1) La sonde et la minuterie ne peuvent pas fonctionner simultanément.

(2) La sonde ne doit pas se réengager avant que le micromètre n'ait été réarmé.

(3) La sonde ne doit pas se désengager en phase de lecture.

(4) La minuterie ne doit pas être réactivée avant que le micromètre n'ait été réarmé.

(5) Le micromètre ne peut être réarmé qu'une fois la lecture prise et la sonde désengagée.

```
Component MicroMeter {

  SubSystem MicroMeter<XW0, const loLmt, hiLmt : MW > {
    State Vector = <Seq, Prb, Tmr>

    Seq : MW; /* Memory word : Reading Sequencer */

    Prb : Jack101 with {
      Σc := { deploy[pe], retract[pd] }
      Σu := { deployend[pee], retractend[pde] }
    }
    Tmr : ODTimerNCNE with {
      Σc := { set[ts], clr[tc] }
      Σu := { exp[te] }
    }
    Synthetic Events := {
      Σc := { clr }
    }

    CtrlSpec {
    /* Implicit */
    /*(1)*/ Prb ∈ {0,1,3} ∧ Tmr ∈ {0,1},
    /* Standard */
```

```
    /*(2)*/ Seq = 2 ∧ Prb = 1,
    /*(3)*/ Seq = 0 ∧ Prb = 3,
    /*(4)*/ Seq ∈ {1,2} ∧ Tmr = 1
    /* Explicit */
    /*(5)    clr <= Seq = 2 */
    }

    f_clr := Seq = 2
    f_pe := Prb = 0 ∧ Seq = 0
    f_pd := Prb = 2 ∧ Seq = 1
    f_ts := Tmr = 0
    f_tc := Tmr = 2 ∧ Seq = 0 ∧ Prb = 2

    Project_te() := {
      if (( XW0 < loLmt ) ∨ (XW0 > hiLmt))
        then return func_fail()
        else return func_pass();
    }

    cycle() := cycle() + {
      func_clr();
    }
  } }
```

D.2.4 PSTester

La composante réutilisable *PSTester* modélise une sonde montée sur un vérin que l'on enfonce dans une perforation pour en tester la profondeur à la sortie de l'usinage. Le test est positif lorsque le vérin atteint la position *complètement déployé* et négatif autrement. Puisqu'il n'existe aucun capteur pour détecter le cas négatif du test, il faut utiliser une minuterie dont l'expiration signale ce cas. La composante comprend donc un vérin dont on peut annuler la commande de déploiement et une minuterie. La figure D.23 en donne le modèle d'interface $(G,\overline{G})$ et la figure D.22 le diagramme d'assemblage.

La caractéristique principale de cette composante est que sa spécification de contrôle est presque complètement *explicite* dans le modèle du diagramme d'assemblage. Les transitions en boucle dans le modèle des instances sont effectivement l'expression de contraintes

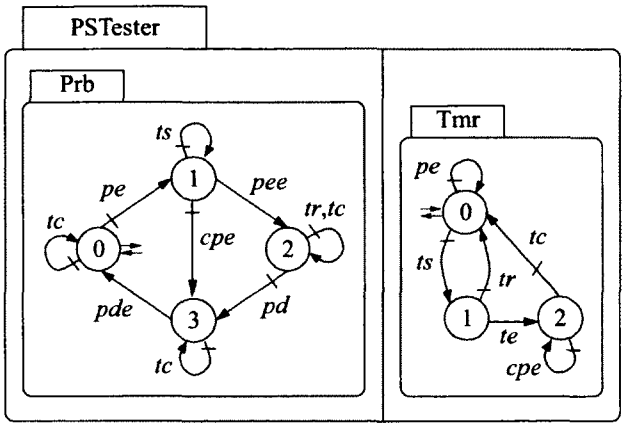


FIGURE D.22 – Composante *PSTester* (diagramme d'assemblage)

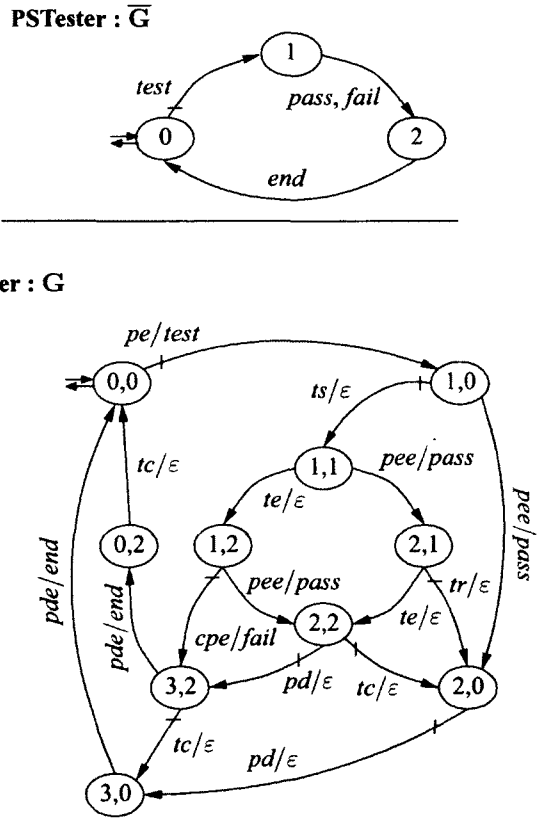


FIGURE D.23 – Composante *PSTester* (sous-système et projection)

que l'on a rendu *explicite* par partage d'événements entre les dispositifs. Ces transitions ont donc été ajoutées à l'instanciation des composantes dans le système dans le but précis d'exprimer des contraintes de contrôle ; elle ne font pas partie des composantes elles-mêmes. Dans ce cas les contraintes sont plus simples à exprimer de cette façon que par l'usage d'une mémoire.

(1) La minuterie ne doit pas être activée lorsque la sonde se désengage ou lorsqu'elle est au repos.

(2) La minuterie ne doit être démarrée (événement *ts*) qu'après que la commande pour engager la sonde ait été donnée.

(3) La minuterie peut être remise à zéro (événement *tr*) lorsque le test s'avère positif.

(4) Dans tous les cas, la minuterie ne peut être réarmée (événement *tc*) qu'une fois un diagnostic de test émis.

(5) Le démarrage d'un test (événement *pe*) ne doit se faire qu'une fois la minuterie réarmée.

(6) L'émission d'un diagnostique d'échec ne peut survenir qu'une fois la minuterie échue.

Cette façon de spécifier les contraintes de contrôle présente le désavantage d'enrichir la structure de transition des composantes d'un sous-système et d'être susceptible de surspécification. Cependant, utilisée de façon prudente, elle s'avère très utile et ne prévient pas nécessairement le parallélisme comme on peut le voir en figure D.23. L'usage d'une mémoire dans ce cas précis requiert de reproduire presque tout le diagramme de G et ne représente aucune économie quand au nombre d'états ou de transitions dans la structure de la solution.

```
Component PSTester {

  SubSystem PSTester {
    State Vector = <Prb, Tmr>

    Prb : Jack111cd with {
      Σc := { deploy[pe], cd[cpe], retract[pd] }
      Σu := { deployend[pee], retractend[pde] }
    }
```

```
    Tmr : ODTimer with {
      Σc := { set[ts], rst[tr], clr[tc] }
      Σu := { exp[te] }
    }

    CtrlSpec {
    /*(1)*/ Prb ∈ {0,3} ∧ Tmr = 1
    /* implicit in DFAs ... */
    /*(2) ts  <= Prb = 1 */
    /*(3) tr  <= Prb = 2 */
    /*(4) tc  <= Prb ∈ {0,2,3} */
    /*(5) pe  <= Tmr = 0 */
    /*(6) cpe <= Tmr = 2 */
    }

    fpe  := Prb = 0 ∧ Tmr = 0
    fcpe := Prb = 1 ∧ Tmr = 2
    fpd  := Prb = 2 ∧ Tmr ∈ {0,2}
    fts  := Tmr = 0 ∧ Prb = 1
    ftr  := Tmr = 1 ∧ Prb = 2
    ftc  := Tmr = 2 ∧ Prb ∈ {0,2,3}

  } }
```

D.2.5 TSCvrDrv

La composante réutilisable *TSCvrDrv* modélise le mécanisme d'entraînement d'un convoyeur avec un temps de latence à l'arrêt. Il se compose d'un entraînement à déclenchement commandé avec arrêt automatique (composante *Stepper*), d'une minuterie et d'un élément de mémoire. La figure D.25 en donne le modèle d'interface (G,$\overline{\text{G}}$) et la figure D.24 le diagramme d'assemblage.

On peut noter dans le code d'intégration qui suit la définition des fonctions $Project_{stp}()$ et $Project_{tc}()$ qui ne sont pas triviales. Étant donnée la définition de la projection, certaines transitions sélectives sur deux événements différents (*stp* et *tc*) dans le sous-système doivent être projetées sur un même événement (*step*) de l'abstraction. Ces définitions peuvent être obtenues directement de la fonction de sortie de la machine de mealy G de la figure D.25

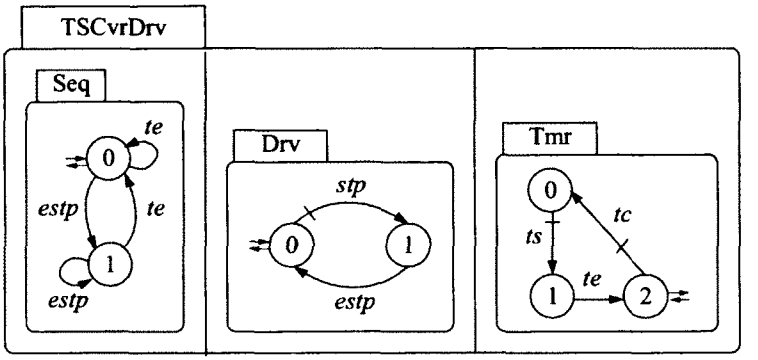


FIGURE D.24 – Composante *TSCvrDrv* (diagramme d'assemblage)

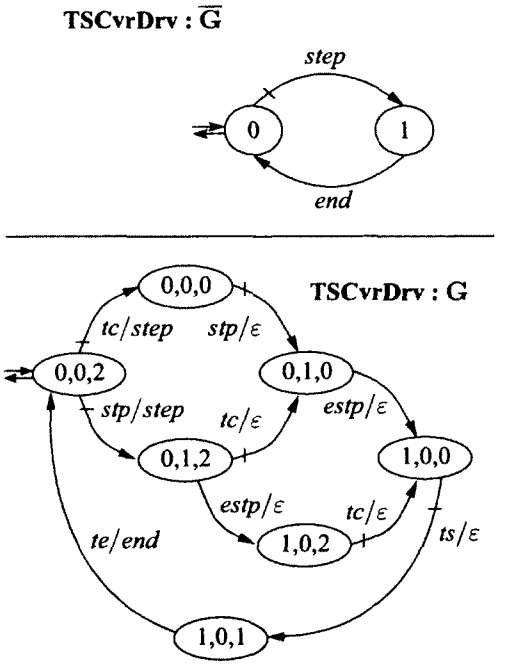


FIGURE D.25 – Composante *TSCvrDrv* (sous-système et projection)

et peuvent donc être générées automatiquement. Quant aux contraintes de la spécification de contrôle toutes sont standards et facile à comprendre et à déduire du modèle.

```
Component TSCvrDrv {
  SubSystem TSCvrDrv {
    State Vector = <Seq, Drv, Tmr>

    Seq : MW; /* Memory word : Sequencer */

    Drv : Stepper with {
      Σc := { start[stp] }
      Σu := { stop[estp] }
    }
    Tmr : ODTimerNCNE with {
      Σc := { set[ts], clr[tc] }
      Σu := { exp[te] }
    }

    Project_stp() := {
      if (Tmr = 2) then return func_step();
      return true;
    }
    Project_tc() := {
      if (Seq = 0 ∧ Drv = 0) then return func_step();
      return true;
    }

    CtrlSpec {
    /*(1)*/ Seq = 0 ∧ Tmr = 1
    /*(2)*/ Seq = 1 ∧ Drv = 1
    }
    f_stp := Drv = 0 ∧ Seq = 0
    f_ts := Tmr = 0 ∧ Seq = 1
    f_tc := Tmr = 2
  } }
```

D.2.6 EHook

La composante EHook (pour *Extensible Hook*) constitue la pince d'une certaine forme de grue. Dans cette forme de grue, la pince (*Hook*) est fixée au bout d'un treuil (*Wch*, pour *Winch* en anglais) pour permettre d'en allonger la portée au moment de saisir ou de déposer un objet (une *pièce*). La composante doit donc définir deux actions : saisir (événement *pu*) et déposer (événement *pd*). Pour EHook, ces actions requièrent toutes les deux d'allonger le treuil *avant* d'effectuer l'action commandée. La figure D.27 en donne le modèle d'interface $(G,\overline{G})$ et la figure D.26 le diagramme d'assemblage. On note qu'ici la pince est instanciée à partir d'une ventouse pneumatique (composante *SuctionCup*) mais qu'elle pourrait tout aussi bien, avec une spécification de contrôle similaire, être instanciée à partir d'un vérin.

Les contraintes, toutes implicites puisque la mémoire n'est présente ici que pour distinguer entre les deux actions de la pince, peuvent se formuler de la façon suivante.

(1) Les commandes du treuil et de la pince doivent être *alternées* de façon stricte.

(2) La pince ne doit pas entrer en action lorsque le treuil est rétracté; elle est alors soit ouverte soit fermée.

(3) En séquence de *prise*, la pince ne peut pas déposer un object et le treuil ne peut pas remonter avec la pince ouverte.

(4) En séquence de *dépôt*, la pince ne peut pas saisir un objet et le treuil ne peut pas remonter avec la pince fermée.

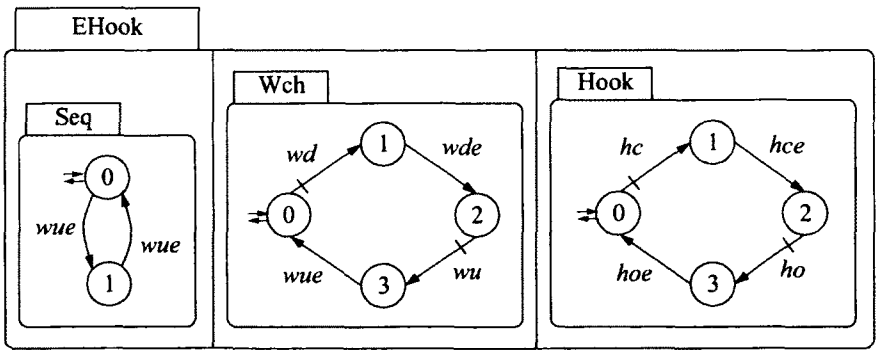


FIGURE D.26 – Composante *EHook* (diagramme d'assemblage)

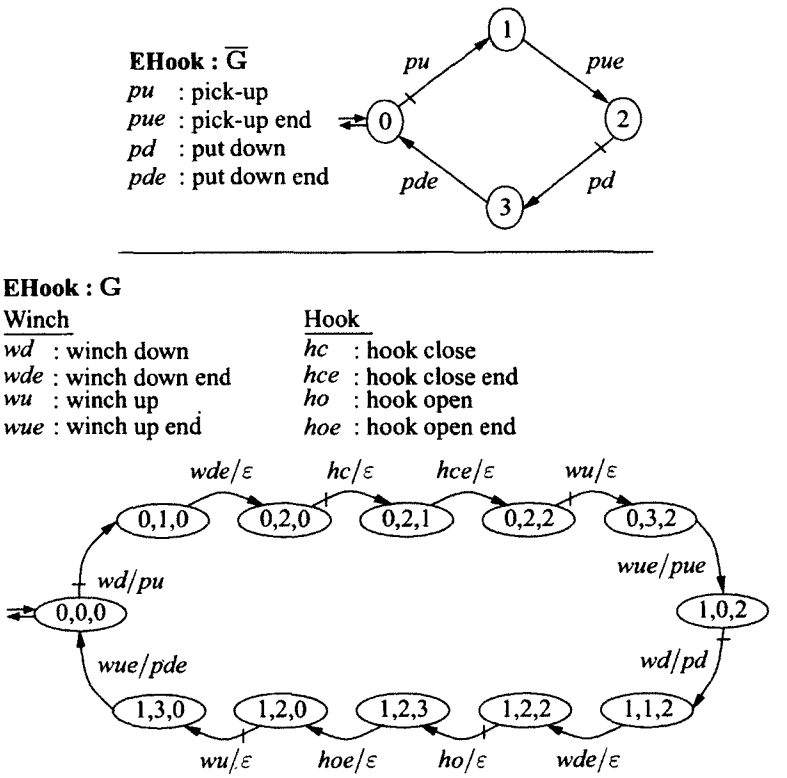


FIGURE D.27 – Composante *EHook* (sous-système et projection)

```
Component EHook {

  SubSystem EHook {
    State Vector = <Seq, Wch, Hook>

    Seq : MW; /* Memory word : Sequencer */

    Wch : Jack111 with {
      Σc := { deploy[wd], retract[wu] }
      Σu := { deployend[wde], retractend[wue] }
    }
    Hook : SuctionCup with {
      Σc := { close[hc], open[ho] }
      Σu := { closeend[hce], openend[hoe] }
    }

    CtrlSpec {
    /* Implicit constraints (forbidden states) */
    /*(1)*/ Wch = {1,3} ∧ Hook ∈ {1,3},
    /*(2)*/ Wch = 0 ∧ Hook ∈ {1,3},
    /*(3)*/
    Seq = 0
    ∧ ((Wch = 2 ∧ Hook = 3) ∨ (Wch = 3 ∧ Hook = 0)),
    /*(4)*/
    Seq = 1
    ∧ ((Wch = 2 ∧ Hook = 1) ∨ (Wch = 3 ∧ Hook = 2))
    }

    fwd := Wch = 0
    fwu :=
      Wch = 2
      ∧ ((Seq = 0 ∧ Hook = 2) ∨ (Seq = 1 ∧ Hook = 0))
    fhc := Hook = 0 ∧ Seq = 0 ∧ Wch = 2
    fho := Hook = 2 ∧ Seq = 1 ∧ Wch = 2

  } }
```

D.2.7 FEHook

La composante FEHook propose de doter la composante EHook d'un mode d'*échec* à la saisie d'une pièce. La figure D.29 en donne le modèle d'interface $(G,\overline{G})$ et la figure D.28 le diagramme d'assemblage.

L'hypothèse d'un échec au dépôt est écartée. À la saisie, si une pièce se trouve véritablement en place, le treuil n'atteindra jamais sa position de pleine extension. Il faut donc amorcer la fermeture de la pince (une ventouse pneumatique) juste après avoir initié la descente du treuil de sorte que la confirmation de fermeture de la pince puisse signaler le succès de l'opération. Si par contre il ne se trouve pas effectivement de pièce en position pour la saisie, le treuil atteindra alors sa position de pleine extension et ceci signale l'absence d'une pièce à saisir et donc un *échec* à la saisie. Dans cette logique, on doit pouvoir annuler la commande d'extension du treuil en cas de succès et annuler la commande de fermeture de la pince en cas d'échec. Il faut donc utiliser une composante `Jack111cd` pour le treuil et une composante `SuctionCupCC` pour la pince. Naturellement il faut aussi enrichir l'espace d'états du séquenceur pour procurer un mode d'échec et récupération à la saisie. Puisqu'il y a plus de commandes à gérer et plus d'information d'états, les requis sont bien sûr plus complexes.

Le mode normal est signalé par `Seq = 0` pour la phase de saisie et `Seq = 2` pour la phase de dépôt. Le mode d'échec et récupération est signalé par `Seq = 1`. L'événement synthétique *clr* ne sert qu'à modifier l'état de la mémoire, à signaler que la phase de récu-

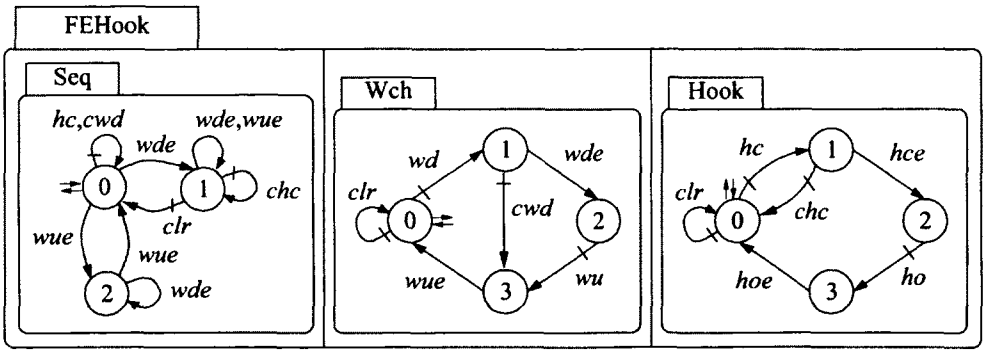
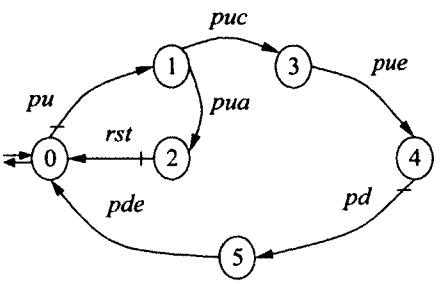


FIGURE D.28 – Composante *FEHook* (diagramme d'assemblage)

FEHook : $\overline{G}$

pu : pick-up
puc : pick-up confirm
pua : pick-up abort
pue : pick-up end
pd : put down
pde : put down end
rst : reset



FEHook : G

Winch	Hook	Synthetic
wd : winch down	*hc* : hook close	*clr* : clear
cwd : cancel winch down	*chc* : cancel hook close	
wde : winch down end	*hce* : hook close end	
wu : winch up	*ho* : hook open	
wue : winch up end	*hoe* : hook open end	

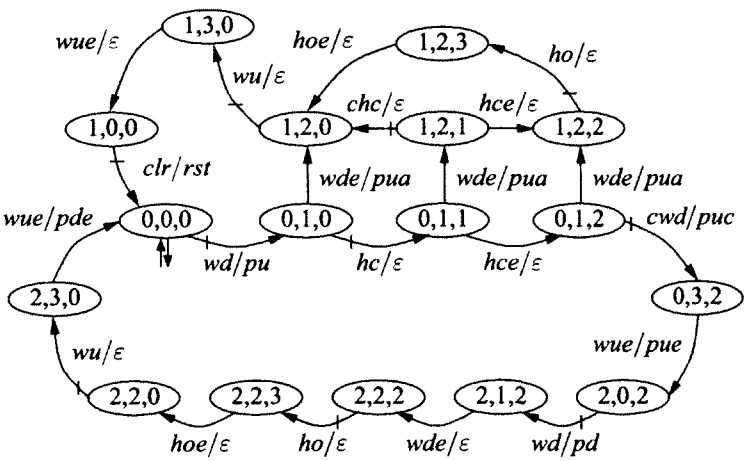


FIGURE D.29 – Composante *FEHook* (sous-système et projection)

pération est terminée et que le treuil et la pince sont à nouveau dans une position adéquate pour le fonctionnement de la composante.

On compte localement 14 requis de contrôle, tous sous forme élémentaire.

(1) La phase de récupération ne se termine que lorsque le treuil est rétracté et la pince ouverte.

(2) La commande de fermeture de la pince n'est disponible qu'en mode de saisie.

(3) La commande d'annulation de déploiement du treuil n'est disponible qu'en mode de saisie.

(4) La commande d'annulation de fermeture de la pince n'est disponible qu'en mode d'échec et récupération.

(5) En mode de récupération le treuil ne doit pas se trouver en déploiement.

(6) La pince ne doit pas entrer en action lorsque le treuil est rétracté ; elle est alors soit ouverte soit fermée.

(7) En mode de saisie, la pince ne peut pas déposer un objet.

(8) En mode de saisie, le treuil ne peut pas remonter avec la pince en fonction (soit en ouverture ou en fermeture).

(9) En mode de saisie, le treuil ne peut pas remonter avec la pince ouverte.

(10) En mode d'échec, on doit ouvrir la pince complètement avant d'entreprendre de relever le treuil.

(11) En mode dépôt, les commandes du treuil et de la pince doivent être *alternées* de façon stricte.

(12) En mode dépôt, le treuil ne peut pas être en déploiement si la pince est ouverte.

(13) En mode dépôt, le treuil ne peut pas être rétracté tant que la pince est fermée.

(14) En mode dépôt, la pince ne peut pas saisir un objet.

On peut noter que l'essentiel des nouveaux requis (par rapport à la spécification de `EHook`) sont occasionnés par l'addition d'un mode d'échec et récupération et à une modification nécessaire pour le mode de saisie (fonctionnement parallèle du treuil et de la pince).

```
Component FEHook {

  SubSystem FEHook {
    State Vector = <Seq, Wch, Hook>

    Seq : MW; /* Memory word : Sequencer */

    Wch : Jack111cd with {
      Σc := { deploy[wd], cd[cwd], retract[wu] }
      Σu := { deployend[wde], retractend[wue] }
    }
    Hook : SuctionCupCC with {
      Σc := { close[hc], cancelclose[chc], open[ho] }
      Σu := { closeend[hce], openend[hoe] }
    }
    Synthetic Events := {
      Σr := { clr }
    }

    CtrlSpec {
    /* Explicit constraints in the model */
    /*(1) clr <= Seq = 1 ∧ Wch = 0 ∧ Hook = 0 */
    /*(2) hc  <= Seq = 0 */
    /*(3) cwd <= Seq = 0 */
    /*(4) chc <= Seq = 1 */
    /* Standard constraints wrt Seq */
    /*(5)*/  Seq = 1 ∧ Wch = 1,
    /* Implicit constraints (forbidden states) */
    /*(6) */ Wch = 0 ∧ Hook ∈ {1,3},
    /*(7) */ Seq = 0 ∧ Wch = 1 ∧ Hook = 3,
    /*(8) */ Seq = 0 ∧ Wch = 3 ∧ Hook ∈ {1,3},
    /*(9) */ Seq = 0 ∧ Wch = 3 ∧ Hook = 0,
    /*(10)*/ Seq = 1 ∧ Wch = 3 ∧ Hook ∈ {1,2,3},
    /*(11)*/ Seq = 2 ∧ Wch ∈ {1,3} ∧ Hook ∈ {1,3},
    /*(12)*/ Seq = 2 ∧ Wch = 1 ∧ Hook = 0,
    /*(13)*/ Seq = 2 ∧ Wch = 3 ∧ Hook = 2,
    /*(14)*/ Seq = 2 ∧ Wch = 2 ∧ Hook = 1
    }
```

f_{clr} := Seq = 1 ∧ Wch = 0 /* => Hook = 0*/
f_{wd} := Wch = 0 ∧ Seq ∈ {0,2} /* Seq ≠ 1*/
f_{cwd} := Wch = 1 ∧ Seq = 0 ∧ Hook = 2
f_{wu} := Wch = 2 ∧ Hook = 0
f_{hc} := Hook = 0 ∧ Wch = 1 /* => Seq = 0 */
f_{chc} := Hook = 1 ∧ Seq = 1
f_{ho} := Hook = 2 ∧ Wch = 2

} }

D.2.8 TOCrane

La composante TOCrane (pour *Two Output Positions Crane*) propose une grue complète avec flèche (*Boom*), pont (*Brg*) et pince extensible (*Wsc* une instance de EHook). La figure D.31 en donne le modèle d'interface ($G,\overline{G}$) et la figure D.30 le diagramme d'assemblage. Cette grue doit aller saisir des pièces à une position d'entrée pour éventuellement les *rejeter* à l'une de ses positions de sortie ou les transférer vers l'autre position de sortie. Aucune interaction nuisible ne peut se produire durant la saisie, le transfert ou le rejet excepté aux extrémités (positions d'entrée et de sortie). La position d'entrée est : flèche à gauche, pont en extension. La position de rejet est : flèche à droite, pont rétracté. La position de transfert est : flèche à droite, pont en extension. Aucune mémoire n'est requise. Tout le contrôle est basé sur les états des dispositifs eux-mêmes donnant lieu à une loi de type SFBC des plus *classique*.

(1) Les commandes de la flèche et du pont doivent être *alternées* de façon stricte.

(2) La flèche ne doit pas être en mouvement lorsque le pont est en extension.

(3) Les commandes de la flèche et de la pince doivent être *alternées* de façon stricte.

(4) La pince ne doit pas déposer une pièce quand la flèche est en position pour saisir (à gauche vers la position d'entrée).

(5) La pince ne doit pas saisir une pièce lorsque la flèche est en position pour déposer (à droite vers la position de sortie).

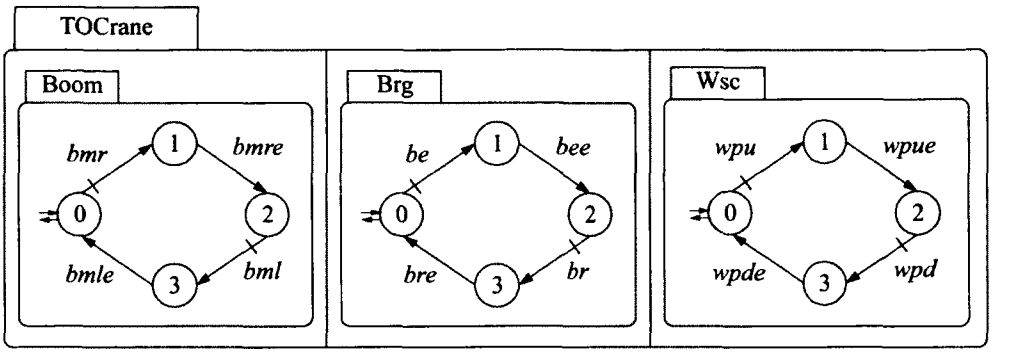
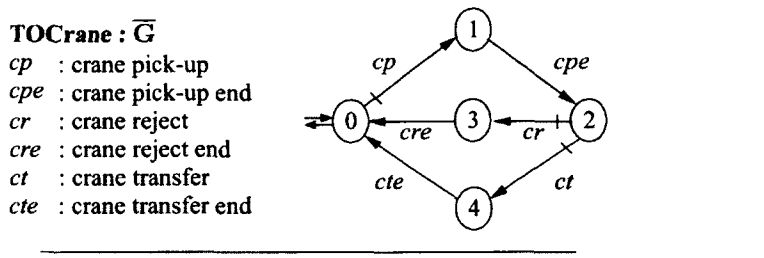


FIGURE D.30 – Composante *TOCrane* (diagramme d'assemblage)



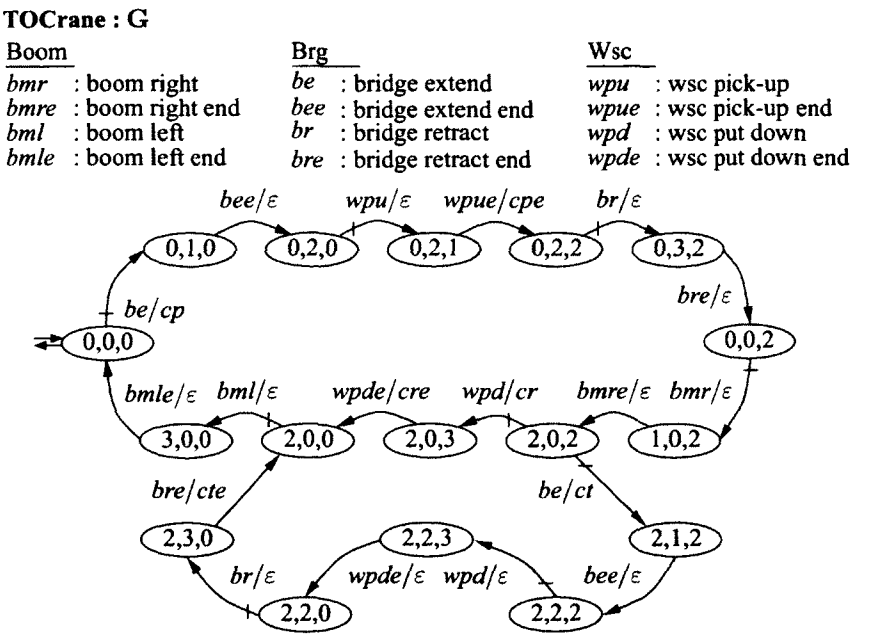


FIGURE D.31 – Composante *TOCrane* (sous-système et projection)

(6) La flèche ne doit pas revenir vers la position d'entrée si la pince est fermée (présumée pleine).

(7) La flèche ne doit pas amorcer de mouvement vers les positions de sortie quand la pince est ouverte (présumée vide).

(8) Les commandes du pont et de la pince doivent être *alternées* de façon stricte.

(9) La pince ne doit pas tenter de saisir une pièce lorsque le pont est rétracté (la position d'entrée se trouvant à l'extrémité c'est-à-dire que le pont doit être en extension pour se trouver au-dessus de la position d'entrée).

(10) Lors d'une manoeuvre de *prise*, le pont ne doit pas se rétracter si la pince est vide (ouverte).

(11) Lors d'une manoeuvre de *prise*, le pont ne doit pas être déployé si la pince est pleine (fermée).

(12) Lors d'un dépôt (rejet ou transfert), le pont ne doit pas être déployé si la pince est vide (ouverte).

(13) Lors d'un dépôt (rejet ou transfert), le pont ne doit pas être rétracté si la pince est pleine (fermée).

```
Component TOCrane {

  SubSystem TOCrane {
    State Vector = <Boom, Brg, Wsc>

    Boom : Jack211a with {
      Σc := { deploy[bmr], retract[bml] }
      Σu := { deployend[bmre], retractend[bmle] }
    }
    Brg : Jack211a with {
      Σc := { deploy[be], retract[br] }
      Σu := { deployend[bee], retractend[bre] }
    }
    Wsc : EHook with {
      Σc := { pu[wpu], pd[wpd] }
      Σu := { pue[wpue], pde[wpde] }
```

```
}

CtrlSpec {
/* All constraints are implicit in the model */
/* Boom x Brg */
/*(1)*/ Boom ∈ {1,3} ∧ Brg ∈ {1,3}
/*(2) No Boom move while Brg extended */
        Boom ∈ {1,3} ∧ Brg = 2
/* Boom x Wsc */
/*(3)*/ Boom ∈ {1,3} ∧ Wsc ∈ {1,3}
/*(4) No Put-down at Boom left */
        Boom = 0 ∧ Wsc = 3
/*(5) No Pick-up at Boom right */
        Boom = 2 ∧ Wsc = 1
/*(6) No left move unless Wsc empty */
        Boom = 3 ∧ Wsc = 2
/*(7) No right move unless Wsc full */
        Boom = 1 ∧ Wsc = 0
/* Brg x Wsc */
/*(8)*/ Brg ∈ {1,3} ∧ Wsc ∈ {1,3}
/*(9) No Pick-up at Brg retracted */
        Brg = 0 ∧ Wsc = 1
/* Boom x Brg x Wsc */
/*(10) On Pick-up, no Brg retract on Wsc empty */
        Boom = 0 ∧ Brg = 3 ∧ Wsc = 0
/*(11) On Pick-up, no Brg extend on Wsc full */
        Boom = 0 ∧ Brg = 1 ∧ Wsc = 2
/*(12) On Put-down, no Brg extend on Wsc empty */
        Boom = 2 ∧ Brg = 1 ∧ Wsc = 0
/*(13) On Put-down, no Brg retract on Wsc full */
        Boom = 2 ∧ Brg = 3 ∧ Wsc = 2
}

f_bmr := Boom = 0 ∧ Brg = 0 ∧ Wsc = 2
f_bml := Boom = 2 ∧ Brg = 0 ∧ Wsc = 0
f_be :=
  Brg = 0
  ∧ ((Boom = 0 ∧ Wsc = 0) ∨ (Boom = 2 ∧ Wsc = 2))
```

f_{br} :=
  Brg = 2
  ∧ ((Boom = 0 ∧ Wsc = 2) ∨ (Boom = 2 ∧ Wsc = 0))
f_{wpu} := Wsc = 0 ∧ Boom = 0 ∧ Brg = 2
f_{wpd} := Wsc = 2 ∧ Boom = 2 ∧ Brg ∈ {0,2}

} }

D.2.9 HSCrane

La composante HSCrane n'est pas à proprement parler une composante de bibliothèque de composantes réutilisables, elle représente plutôt une solution partielle à un problème de conception. La figure D.33 en donne le modèle d'interface ($G,\overline{G}$) et la figure D.32 le diagramme d'assemblage.

On peut parler d'une composante locale que l'on peut définir à l'intérieur d'un problème plus complexe (ici le problème développé au chapitre 5). Cette composante est en fait une simplification de l'interface de TOCrane qui tient compte du fait que dans le problème du chapitre 5 il n'y a pas de possibilité d'interactions nuisibles aux positions de sortie mais qu'il y a une interaction potentiellement nuisible à la position d'entrée pendant que la pince est en train de saisir une pièce. Il suffit donc de savoir quand cette phase se termine (événement *pue* pour *pick-up end*) pour éviter les problèmes. Aussi, la décision de transférer ou de rejeter est rendue explicite par l'usage d'une mémoire de sélection d'opération et la décision est prise dès le démarrage de la grue puisqu'alors l'information nécessaire est déjà disponible.

(1) Une commande de rejet ne peut être donnée que si la grue est à l'arrêt (Sel = 0).

(2) Une commande de transfert ne peut être donnée que si la grue est à l'arrêt (Sel = 0).

(3) La grue ne doit pas saisir une pièce tant qu'une commande (transfert ou rejet) n'a pas été donnée (Sel = 0).

(4) Un rejet ou un transfert requiert une commande.

(5) La grue ne peut pas transférer en mode rejet (Sel = 1).

(6) La grue ne peut pas rejeter en mode transfert (Sel = 2).

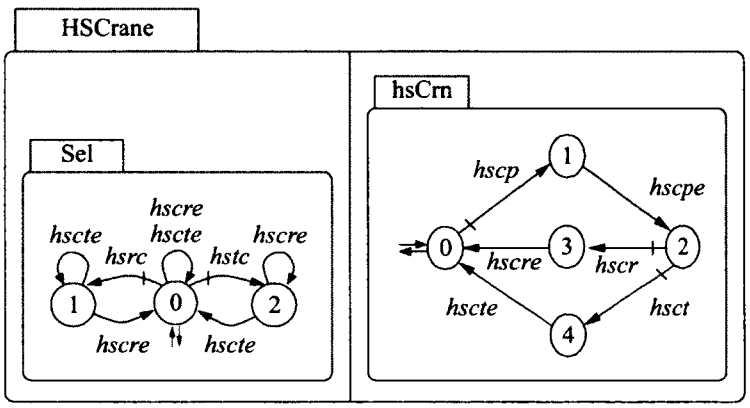


FIGURE D.32 – Composante *HSCrane* (diagramme d'assemblage)

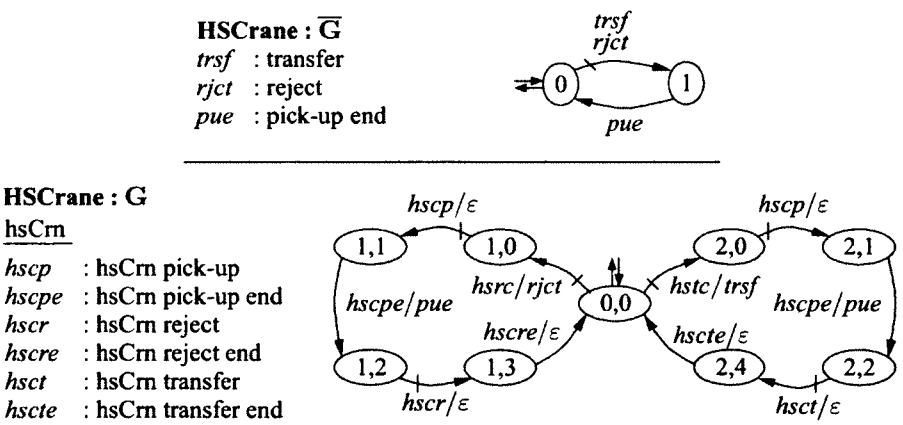


FIGURE D.33 – Composante *HSCrane* (sous-système et projection)

```
Component HSCrane {

  SubSystem HSCrane {
    State Vector = <Sel, hsCrn>

    Sel : MW; /* Command selector memory */

    hsCrn : TOCrane with {
      Σc := { cp[hscp], cr[hscr], ct[hsct] }
      Σu := { cpe[hscpe], cre[hscre], cte[hscte] }
    }
    Synthetic Events := {
      Σc := { hstc, hsrc }
    }

    CtrlSpec {
      (explicit)
      /*(1)*/ hsrc <= Sel = 0
      /*(2)*/ hstc <= Sel = 0
      (Implicit)
      /*(3) Cannot pickup without a command */
        Sel = 0 ∧ hsCrn = 1
      (standard)
      /*(4) Cannot transfer or reject without a command */
        Sel = 0 ∧ hsCrn ∈ {3,4}
      /*(5) Cannot transfer in reject mode */
        Sel = 1 ∧ hsCrn = 4
      /*(6) Cannot reject in transfer mode */
        Sel = 2 ∧ hsCrn = 3
    }

    fhstc := Sel = 0
    fhsrc := Sel = 0
    fhscp := hsCrn = 0 ∧ Sel ∈ {1,2}
    fhscr := hsCrn = 2 ∧ Sel = 1
    fhsct := hsCrn = 2 ∧ Sel = 2

  } }
```

D.2.10 FHTOCrane

La composante `FHTOCrane` propose un raffinement sur la grue de type `TOCrane` utilisant une pince dotée d'un mode d'échec et récupération lors de la *prise* d'une pièce. La figure D.35 en donne le modèle d'interface ($G,\overline{G}$) et la figure D.34 le diagramme d'assemblage.

La description de la grue de type `TOCrane` s'applique intégralement ici en ce qui concerne les positions d'entrée et de sortie et à la logique de rejet et de transfert. La seule différence est qu'il faut gérer un mode d'échec lors de la saisie d'une pièce et à cette fin il est nécessaire d'ajouter une mémoire. Les requis aussi sont (étonnamment) assez semblables. Seul le requis (10) doit être altéré et le requis (14) ajouté pour tenir compte du nouveau mode d'échec.

1. Les commandes de la flèche et du pont doivent être *alternées* de façon stricte.
2. La flèche ne doit pas être en mouvement lorsque le pont est en extension.
3. Les commandes de la flèche et de la pince doivent être *alternées* de façon stricte.
4. La pince ne doit pas déposer une pièce quand la flèche est en position pour saisir (à gauche vers la position d'entrée).
5. La pince ne doit pas saisir une pièce lorsque la flèche est en position pour déposer (à droite vers la position de sortie).
6. La flèche ne doit pas revenir vers la position d'entrée si la pince est fermée (présumée pleine).

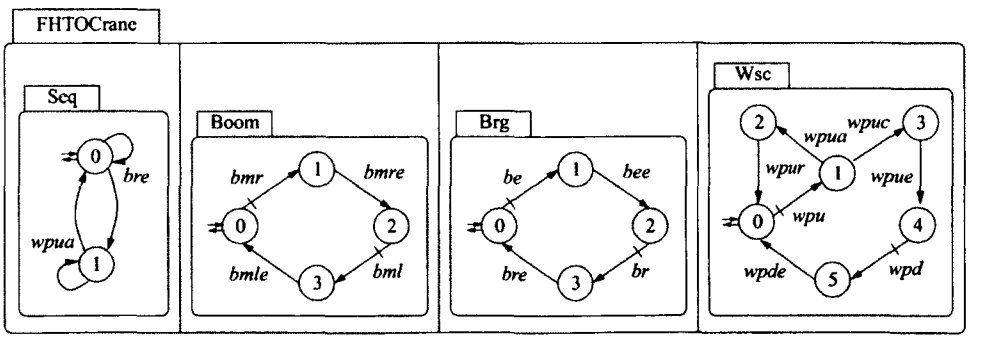
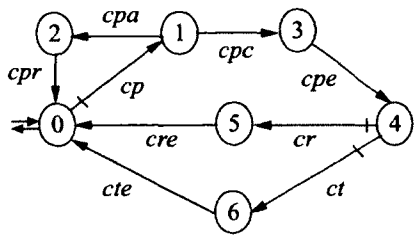


FIGURE D.34 – Composante *FHTOCrane* (diagramme d'assemblage)

FHTOCrane : $\overline{G}$

cp : crane pick-up
cpa : crane pick-up abort
cpc : crane pick-up confirm
cpr : cpa recovered
cpe : crane pick-up end
cr : crane reject
cre : crane reject end
ct : crane transfer
cte : crane transfer end



FHTOCrane : G

Boom

bmr : boom right
bmre : boom right end
bml : boom left
bmle : boom left end

Brg

be : bridge extend
bee : bridge extend end
br : bridge retract
bre : bridge retract end

Wsc

wpu : pick-up
wpua : pick-up abort
wpuc : pick-up confirm
wpur : pick-up recover
wpue : pick-up end
wpd : put down
wpde : put down end

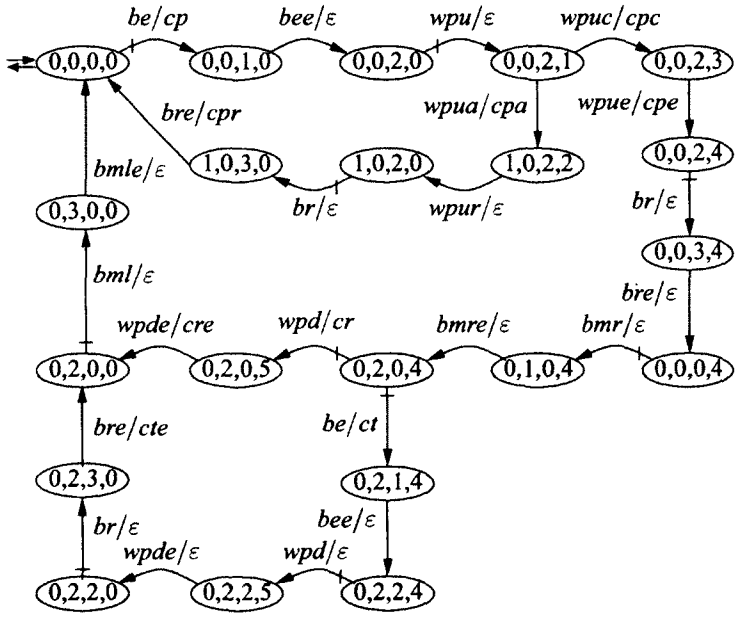


FIGURE D.35 – Composante *FHTOCrane* (sous-système et projection)

7. La flèche ne doit pas amorcer de mouvement vers les positions de sortie quand la pince est ouverte (présumée vide).
8. Les commandes du pont et de la pince doivent être *alternées* de façon stricte.
9. La pince ne doit pas tenter de saisir une pièce lorsque le pont est rétracté (la position d'entrée se trouvant à l'extrémité c'est-à-dire que le pont doit être en extension pour se trouver au-dessus de la position d'entrée).
10. Lors d'une manoeuvre de *prise*, en mode normal (c'est-à-dire `Seq = 0`), le pont ne doit pas se rétracter si la pince est vide (ouverte).
11. Lors d'une manoeuvre de *prise*, le pont ne doit pas être déployé si la pince est pleine (fermée).
12. Lors d'un dépôt (rejet ou transfert), le pont ne doit pas être déployé si la pince est vide (ouverte).
13. Lors d'un dépôt (rejet ou transfert), le pont ne doit pas être rétracté si la pince est pleine (fermée).
14. En mode d'échec, la pince ne doit pas tenter de saisir une pièce à nouveau.

```
Component FHTOCrane {

  SubSystem FHTOCrane {
    State Vector = <Seq, Boom, Brg, Wsc>

    Seq : MW; /* Mode : normal(0), pick-up abort(1) */

    Boom : Jack211a with {
      Σc := { deploy[bmr], retract[bml] }
      Σu := { deployend[bmre], retractend[bmle] }
    }
    Brg : Jack211a with {
      Σc := { deploy[be], retract[br] }
      Σu := { deployend[bee], retractend[bre] }
    }
    Wsc : FEHook with {
      Σc := { pu[wpu], pd[wpd] }
      Σu := { pua[wpua], puc[wpuc], pur[wpur], pue[wpue],
```

```
            pde[wpde] }
}

CtrlSpec {
/* Constraints implicit in the model */
/* Boom x Brg */
/*(1)*/ Boom ∈ {1,3} ∧ Brg ∈ {1,3}
/*(2) No Boom move while Brg extended */
        Boom ∈ {1,3} ∧ Brg = 2
/* Boom x Wsc */
/*(3)*/ Boom ∈ {1,3} ∧ Wsc ∈ {1,2,3,5}
/*(4) No Put-down at Boom left */
        Boom = 0 ∧ Wsc = 5
/*(5) No Pick-up at Boom right */
        Boom = 2 ∧ Wsc = 1
/*(6) No left move on full Wsc */
        Boom = 3 ∧ Wsc = 4
/*(7) No right move on empty Wsc */
        Boom = 1 ∧ Wsc = 0
/* Brg x Wsc */
/*(8)*/ Brg ∈ {1,3} ∧ Wsc ∈ {1,2,3,5}
/*(9) No Pick-up at Brg retracted */
        Brg = 0 ∧ Wsc = 1
/* Seq x Boom x Brg x Wsc */
/*(10) No Brg retract on empty Wsc in normal mode */
        Seq = 0 ∧ Boom = 0 ∧ Brg = 3 ∧ Wsc = 0
/* Boom x Brg x Wsc */
/*(11) No Brg extend on full Wsc when at left */
        Boom = 0 ∧ Brg = 1 ∧ Wsc = 4
/*(12) No Brg extend on empty Wsc when at right */
        Boom = 2 ∧ Brg = 1 ∧ Wsc = 0
/*(13) No Brg retract on full Wsc when at right */
        Boom = 2 ∧ Brg = 3 ∧ Wsc = 4
/* Standard constraints wrt Seq */
/*(14) No pick-up in failure mode */
        Seq = 1 ∧ Wsc = 1
}
```

```
    f_bmr := Boom = 0 ∧ Brg = 0 ∧ Wsc = 4
    f_bml := Boom = 2 ∧ Brg = 0 ∧ Wsc = 0
    f_be :=
      Brg = 0
      ∧ ((Boom = 0 ∧ Wsc = 0) ∨ (Boom = 2 ∧ Wsc = 4))
    f_br :=
      Brg = 2 ∧
      (   Seq = 1 ∧ Wsc = 0
        ∨ Seq = 0 ∧ Boom = 0 ∧ Wsc = 4
        ∨ Seq = 0 ∧ Boom = 2 ∧ Wsc = 0
      )
    f_wpu := Wsc = 0 ∧ Seq = 0 ∧ Boom = 0 ∧ Brg = 2
    f_wpd := Wsc = 4 ∧ Boom = 2 ∧ Brg ∈ {0,2}

  } }
```

D.2.11 FHHSCrane

La composante FHHSCrane reprend, par rapport à la composante FHTOCrane, le même exercice que celui fait pour HSCrane relativement à la composante TOCrane, et permet de vérifier que la même simplification d'interface est possible. La figure D.37 en donne le modèle d'interface ($G,\overline{G}$) et la figure D.36 le diagramme d'assemblage.

Le même type de mémoire de sélection est nécessaire. Étonnamment, l'ensemble des requis est plus simple et plus clair.

1. Une commande de rejet ne peut être donnée que si la grue est à l'arrêt (Sel = 0).
2. Une commande de transfert ne peut être donnée que si la grue est à l'arrêt (Sel = 0).
3. La grue ne doit pas tenter de saisir, rejeter ou transférer une pièce tant qu'une commande (transfert ou rejet) n'a pas été donnée (Sel = 0).
4. La grue ne peut pas transférer en mode rejet (Sel = 2).
5. La grue ne peut pas rejeter en mode transfert (Sel = 1).

```
Component FHHSCrane {

  SubSystem FHHSCrane {
```

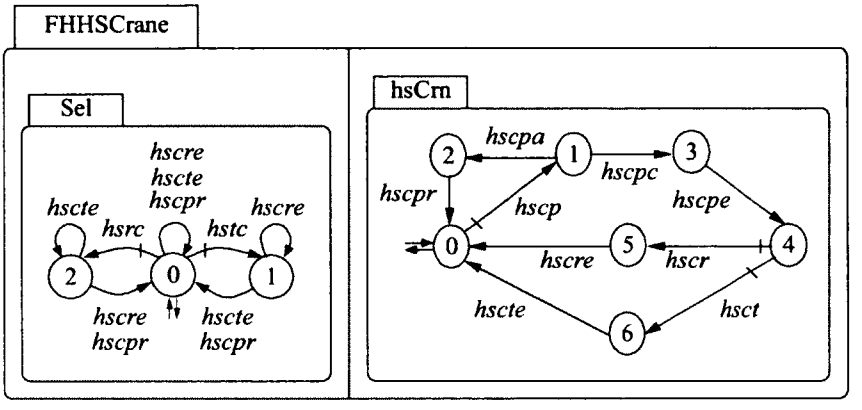
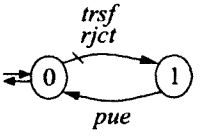


FIGURE D.36 – Composante *FHHSCrane* (diagramme d'assemblage)

FHHSCrane : $\overline{G}$
trsf : transfer
rjct : reject
pue : pick-up end



FHHSCrane : G
hsCrn

hscp : hsCrn pick-up
hscpa : hsCrn pick-up abort
hscpc : hsCrn pick-up confirm
hscpr : hsCrn pick-up reset
hscpe : hsCrn pick-up end
hscr : hsCrn reject
hscre : hsCrn reject end
hsct : hsCrn transfer
hscte : hsCrn transfer end

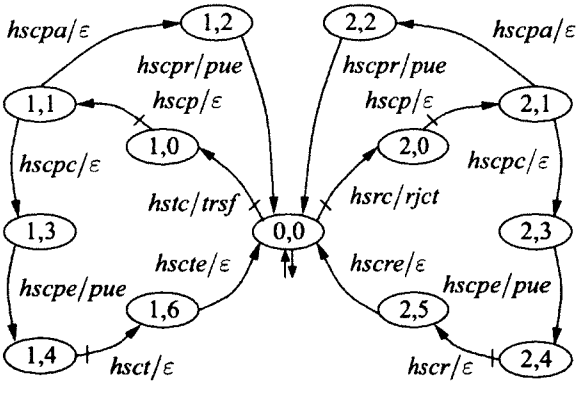


FIGURE D.37 – Composante *FHHSCrane* (sous-système et projection)

```
State Vector = <Sel, hsCrn>

Sel : MW; /* Command selector memory */

hsCrn : FHTOCrane with {
  Σc := { cp[hscp], cr[hscr], ct[hsct] }
  Σu := { cpa[hscpa], cpc[hscpc], cpr[hscpr],
          cpe[hscpe], cre[hscre], cte[hscte] }
}
Synthetic Events := {
  Σc := { hstc, hsrc }
}

CtrlSpec {
  (explicit)
  /*(1)*/ hsrc <= Sel = 0
  /*(2)*/ hstc <= Sel = 0
  (standard)
  /*(3) Cannot pickup-up transfer or reject
        without a command */
    Sel = 0 ∧ hsCrn ∈ {2,5,6}
  /*(4) Cannot transfer in reject mode */
    Sel = 2 ∧ hsCrn = 6
  /*(5) Cannot reject in transfer mode */
    Sel = 1 ∧ hsCrn = 5
}

fhstc := Sel = 0
fhsrc := Sel = 0
fhscp := hsCrn = 0 ∧ Sel ∈ {1,2}
fhscr := hsCrn = 4 ∧ Sel = 2
fhsct := hsCrn = 4 ∧ Sel = 1

} }
```

D.3 Sommaire

Pour compléter le catalogue il est intéressant de tenter de faire ressortir, au moyen de quelques statistiques, les points forts ou les faiblesses de la métaphore de composantes réutilisables et de la méthode de conception utilisée. À cette fin il est possible d'utiliser deux *métriques* :

1. la complexité des sous-systèmes conçus sans abstraction (par rapport aux sous-systèmes conçus à partir d'abstractions) ;
2. le nombre de requis récupérés au moment de l'utilisation d'une composante réutilisable.

Le tableau D.1 liste les statistiques pour l'ensemble des composantes élémentaires du catalogue et le tableau D.2 liste les statistiques relatives aux composantes mixtes. D'abord il est important de comprendre ce que veulent dire ces métriques et comment sont établies ces statistiques.

À la colonne ***Espace d'états (exhaustif)***, on retrouve le nombre d'états des modèles tels qu'on les utilises dans une conception basée sur le contexte original de SCT (contexte de Ramadge et Wonham). Cette statistique est fréquemment utilisée dans la littérature sur le sujet et constitue un bon indicateur de complexité autant pour la conception que pour la synthèse. Par exemple pour la composante élémentaire `Jack211a` (tableau D.1) on a 9 états si on élimine les configurations de capteurs et d'actionneur qui ne peuvent pas se présenter. Bien sûr le dispositif doit d'abord être *préconditionné*, sinon aucune synthèse n'est possible, et le résultat est un modèle à 6 états (colonne ***Espace d'états (sous-système)***). Mais il est nécessaire de tenir compte du modèle exhaustif pour la mesure de complexité même si un travail *ad hoc* doit d'abord être réalisé pour pouvoir éventuellement utiliser la synthèse SCT. Ce travail fait effectivement partie de l'effort de modélisation.

Pour les composante mixtes (tableau D.2), la statistique ***Espace d'états (sous-système)*** représente le produit du nombre d'états présents dans l'interface des instances utilisées dans la composition du sous-système. Par exemple la composante `TOCrane`, utilise trois instances de composantes réutilisables comportant chacune quatre états dans leur interface. Le sous-système de `TOCrane` comporte donc 64 (c'est-à-dire $4 \times 4 \times 4$) états. Il faut se rappeler que ces composantes sont disjointes (elles ne partagent *a priori* aucun de leurs

TABLE D.1 – Complexité des composantes élémentaires

Composante	*Nombres de requis*	*Espace d'états (interface)*	*Espace d'états (sous-système)*	*Espace d'états (exhaustif)*
jack111	2	4	6	6
jack111SS	2	2	6	6
jack111cdcr	4	4	6	6
jack111cd	3	4	6	6
jack111cr	3	4	6	6
jack110	2	4	6	8
jack101	2	4	6	8
jack100	2	4	6	8
jack211a	2	4	6	9
jack211b	2	4	6	9
ADRelay	2	4	6	8
SDRelay	2	4	4	4
NDRelay	2	1	4	4
ODTimer	3	3	4	4
Injector	2	4	8	12
Stepper	2	2	3	4
SuctionCup	2	4	4	6
SuctionCupCC	2	4	4	6

TABLE D.2 – Complexité des composantes mixtes

Composante	*Nombres de requis*	*Espace d'états (interface)*	*Espace d'états (sous-système)*	*Espace d'états (exhaustif)*
LRCrane	4 (+4)	6	16	54
BLRCrane	4 (+4)	9	16	54
Drill	10 (+6)	2	128	864
Micrometer	5 (+5)	4	36	96
PSTester	6 (+6)	3	12	24
TSCvrDrv	2 (+5)	2	12	32
EHook	6 (+4)	4	32	72
FEHook	14 (+5)	6	48	108
TOCrane	13 (+14)	5	64	5 832
HSCrane	6 (+27)	2	15	17 496
FHTOCrane	14 (+23)	7	192	17 496
FHHSCrane	5 (+37)	2	21	52 488

événements). Leur composition parallèle comporte donc le même nombre d'états que le produit cartésien de leurs espaces d'états respectifs. De façon analogue, la statistique ***Espace d'états (exhaustif)*** est obtenue en faisant le produit du nombre d'états des modèles exhaustifs des instances constituantes. Dans l'exemple `TOCrane` on aura $9 \times 9 \times 48 = 3\,888$ états au modèle exhaustif puisque `TOCrane` est constituée de deux `Jack211a` (à 9 états pour le modèle exhaustif) et de un `EHook` (à 48 états pour le modèle exhaustif). Ce faisant on peut remarquer que `EHook` contient une mémoire, et que cette mémoire est incluse dans le modèle exhaustif de `EHook`. Ne gonfle-t-on pas alors artificiellement la statistique ?

La réponse est non. Il faut se rappeler que les calculs de synthèse effectués avec cette méthode de conception ont pour but d'obtenir une loi de contrôle de type SFBC ; plus précisément, en général, un DSFBC (voir [62]) c'est-à-dire une SFBC contenant de la mémoire. Il faut donc disposer d'un espace d'états suffisant pour exprimer les requis de contrôle sous forme de familles d'états proscrits (spécification par évitement d'états). La mémoire ajoutée en cours de modélisation n'a pour but que de permettre ce genre de spécification, c'est-à-dire d'accroître l'espace des états du système. C'est la notion de *mémoire adéquate* à la section 7.6 du chapitre 7 de [62]. Cette *mémoire adéquate*, bien qu'elle partage tous ses événements avec le système, n'impose aucune contrainte au système *a priori*. Elle peut donc avoir le même effet sur la taille de l'espace d'états résultant qu'une composante disjointe ayant la même taille et parfois pire. Il est impossible de la négliger.

La statistique ***Espace d'états (interface)*** donne une idée des économies réalisées par l'abstraction.

La statistique ***Nombres de requis*** (établie en nombre de requis élémentaires tels que définis en préambule de cette annexe) donne une idée de la complexité encapsulée par abstraction. Réutiliser une composante de type `FHHSCrane` (voir tableau D.2), par exemple, évite d'avoir à formuler 43 requis de contrôle : 5 requis directs (dans la définition de cette composante) et 38 requis indirects, inclus par l'ensemble des instances de composantes réutilisables qui constituent le sous-système de `FHHSCrane`.

Finalement on peut noter qu'aucune des synthèses requises pour élaborer les composantes de cet inventaire n'a demandé plus de 3 dixièmes de seconde avec le logiciel *Supremica* [2]. Ce résultat ne doit pas surprendre puisque la méthode encourage la modularisation des requis par abstraction, donc la synthèse se fait généralement sur des sous-

systèmes de tailles modestes. Aussi, plus surprenant, tous les calculs pour obtenir des modèles d'*observateurs* (algorithme de Wong [60]) n'ont requis que quelques dixièmes de secondes. Élaborer la projection est un processus heuristique plus long qui requiert d'émettre des hypothèses et de refaire les calculs pour obtenir les observateurs correspondants plusieurs fois. Mais les calculs eux-mêmes ne semblent pas *a priori* en être le facteur limitatif. Il est beaucoup plus difficile, par exemple, de se donner une idée des hypothèses à formuler pour un modèle comportant 500 à 1 000 états. C'est là, certainement, un facteur limitatif.

De cette courte analyse il ressort deux points importants. Bien que l'abstraction en elle-même permette des économies d'échelle pour la synthèse, les principaux atouts des composantes réutilisables semblent se situer au niveau de la modularisation des requis de contrôle. Premièrement, la modularisation des requis de contrôle et leur encapsulation par abstraction permet la réutilisation directe d'un travail difficile et long à faire (l'élaboration de bons requis de contrôle). Deuxièmement, cette modularisation permet d'approcher les problèmes de contrôle de façon structurée, par étape, et de s'attacher à chaque étape à spécifier et tester des ensembles de requis relativement modestes.

La station d'usinage du *MPS* peut servir d'illustration à ce dernier point. Cette station comprend :

- un vérin à l'admission (composante `Jack100`, modèle exhaustif à 8 états et interface à 2 états) ;
- un convoyeur temporisé pour l'entraînement (`TSCvrDrv`, 16 et 2 états) ;
- une perceuse (`Drill`, 864 et 2 états) ;
- une sonde de contrôle de qualité (`PSTester`, 36 et 3 états) ;
- une grue de transfert (`FHHSCrane`, 34 992 et 2 états)
- un registre à décalage de 4 positions pour modéliser l'état des positions d'usinage (16 états) ;
- un registre à décalage de 2 positions pour transporter le résultat du test de qualité (4 états).

Le modèle exhaustif comporte $8{,}916 \times 10^{12}$ états contre 3 072 pour le modèle abstrait.

Le modèle abstrait de la station d'usinage suggère une approche en deux étapes : d'abord établir les requis de sécurité pour le matériel (modèle abstrait de 48 états), ensuite ajouter la mémoire et les requis qui lui sont relatifs. La mémoire peut être elle même traitée

en deux étapes ; un registre à décalage à la fois. Sur le modèle exhaustif la situation est plus embrouillée dû à la taille du modèle. Déjà la taille du modèle du matériel est considérable, même en excluant la grue à prime abord, plus de trois millions d'états. Dans ces conditions il est difficile d'établir une stratégie pour l'analyse et la solution du problème. Même si on y parvient, avec des modèles d'une telle taille, il devient vite difficile d'émettre des hypothèses quant aux requis et d'en analyser l'effet sur le modèle du problème. En d'autre termes, les requis spécifiés doivent se révéler *parfaits du premier coup* car il n'y a pas de moyen d'en vérifier l'effet sauf par la synthèse ; un diagnostique de type tout ou rien.

Assez clairement la modularisation des requis de contrôle, procurée par l'usage de composantes réutilisables, semble être nécessaire à la formulation d'une méthode de résolution structurée de problèmes de contrôle d'envergure non triviale. Bien sûr la présence de composantes réutilisables permet aussi de limiter l'envergure du travail de modélisation *ad hoc* nécessaire dans l'élaboration de problème de contrôle. Ce travail occupe aujourd'hui une importance disproportionnée dans toutes les variantes de SCT et limite en général la taille, et parfois même l'intérêt, des problèmes considérés.

Bibliographie

[1] J.-R. ABRIAL. *The B-Book : Assigning Programs to Meanings*. Cambridge University Press, Cambridge, UK, 1996.

[2] K. ÅKESSON, M. FABIAN, H. FLORDAL et R. MALIK. « Supremica – An integrated environment for verification, synthesis and simulation of discrete event systems ». Dans *Proc. 8th International Workshop on Discrete Event Systems*, pages 384–385, Ann Arbor, Michigan, 2006.

[3] A. ARNOLD. *Systèmes de transitions finis et sémantique des processus communiquants*. Masson, Paris, 1992.

[4] G. BEHRMANN, A. DAVID et K. G. LARSEN. « A Tutorial on Uppaal ». Rapport technique, Department of Computer Science, Aalborg University, Aalborg, Danemark, 2004.

[5] E. BRINKSMA et A. MADER. « Verification and optimization of a PLC control schedule ». Dans K. HAVELUND, J. PENIX et W. VISSER, éditeurs, *Proceedings of the 7th International SPIN Workshop on SPIN Model Checking and Software Verification*, volume 1885 de *Lecture Notes in Computer Sciences*, pages 73–92. Springer-Verlag Berlin Heidelberg, 2000.

[6] G. CIARDO, G. LÜTTGEN et R. SIMINICEANU. « Saturation : An Efficient Iteration Strategy for Symbolic State-space Generation ». ICASE Report 2001-5, NASA, 2001.

[7] D. CÔTÉ. « Système de génération de code pour des usines industrielles modulaires ». Mémoire de maîtrise, Université de Sherbrooke, Sherbrooke, Canada, 2004.

[8] D. CÔTÉ, R. ST-DENIS et S. KERJEAN. « Generative programming for programmable logic controllers ». Dans *Proc. 10th IEEE Conference on Emerging Technologies and Factory Automation ETFA 2005*, volume 2, pages 741–748, Catania, Italie, 2005.

[9] P. DAI. « Synthesis Method for Hierarchical Interface-based Supervisory Control ». Mémoire de maîtrise, McMaster University, Hamilton, Canada, 2006.

[10] M. H. DE QUEIROZ et J. E. R. CURY. « Modular supervisory control of large scale discrete-event systems ». Dans *Proc. WODES'00, Discrete Event Systems : Analysis and Control*, pages 103–110, Ghent, Belgique, 2000.

[11] M. H. DE QUEIROZ et J. E. R. CURY. « Synthesis and implementation of local modular supervisory control for a manufacturing cell ». Dans *Proc. of the Sixth International Workshop on Discrete Event Systems*, pages 377–382, Zaragoza, Spain, 2002.

[12] R. DRECHSLER et D. SIELING. « Binary decision diagrams in theory and practice ». *International Journal on Software Tools for Technology Transfer*, 3(2) :112–136, 2001.

[13] G. EKBERG et B. H. KROGH. « Programming discrete control systems using state machine templates ». Dans *Proc. of the 8th International Workshop on Discrete Event Systems*, pages 194–200, Ann Arbor, Michigan, 2006.

[14] L. FENG. « *Computationally Efficient Supervisor Design for Discrete-Event Systems* ». Thèse de doctorat, University of Toronto, Toronto, Canada, 2007.

[15] L. FENG et W. M. WONHAM. « On the computation of natural observers in discrete-event systems ». *Discrete Event Dynamic Systems*, 20(1) :20–63, 2010.

[16] J.-C. FERNANDEZ. « An implementation of an efficient algorithm for bisimulation equivalence ». *Science of Computer Programming*, 13(2-3) :219–236, 1990.

[17] FESTO. « *Processing Station–Modular Production System* ». Festo Didactic GmbH & Co., Denkendorf, 1998.

[18] FESTO. « *Sorting Station–Modular Production System* ». Festo Didactic GmbH & Co., Denkendorf, 1998.

[19] FESTO. « *Testing Station–Modular Production System* ». Festo Didactic GmbH & Co., Denkendorf, 1998.

[20] FESTO. « *Distribution Station–Modular Production System* ». Festo Didactic GmbH & Co., Denkendorf, 1999.

[21] FESTO. « *Handling Station–Modular Production System* ». Festo Didactic GmbH & Co., Denkendorf, 1999.

[22] G. FREY et L. LITZ. « Formal methods in PLC programming ». Dans *Proc. of IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 2431–2436, Nashville, TN, 2000.

[23] B. GAUDIN. « *Synthèse de contrôleurs sur des systèmes à événements discrets structurés* ». Thèse de doctorat, Université de Rennes I, Rennes, France, 2004.

[24] P. GOHARI et W. M. WONHAM. « A linguistic framework for controlled hierarchical DES ». Dans *Proc. of the 4th International Workshop on Discrete Event Systems*, pages 207–212, Reims, France, 1998.

[25] G. GÖSSLER et J. SIFAKIS. « Composition for component-based modeling ». *Science of Computer Programming*, 55(1-3) :161–183, 2005.

[26] J. GUNNARSSON. « *Symbolic Methods and Tools for Discrete Event Dynamic Systems* ». Thèse de doctorat, Linköping University, Division of Automatic Control, Department of Electrical Engineering, 1997.

[27] D. HAREL et M. POLITI. *Modeling Reactive Systems with Statecharts*. McGraw-Hill Books, New-York, 1998.

[28] J. HUANG et R. KUMAR. « Optimal nonblocking directed control of discrete event systems ». *IEEE Transactions on Automatic Control*, 53(7) :1592–1603, 2008.

[29] P. HUBBARD et P. E. CAINES. « Dynamical consistency in hierarchical supervisory control ». *IEEE Transactions on Automatic Control*, 47(1) :37–52, 2002.

[30] T. KAM, T. VILLA, R. BRAYTON et A. SANGIOVANNI. « Multi-valued decision diagrams : theory and applications ». *International Journal on Multi-Valued Logic*, 4(1-2) :9–62, 1998.

[31] Zvi KOHAVI. *Switching and Finite Automata Theory*. McGraw-Hill, 1970.

[32] R. KUMAR et V. K. GARG. *Modeling and Control of Logical Discrete Event Systems*. Kluwer Academic Publishers, Boston, 1995.

[33] R. J. LEDUC. « PLC Implementation of a DES Supervisor for a Manufacturing Testbed : An Implementation Perspective ». Mémoire de maîtrise, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, 1996.

[34] R. J. LEDUC. « *Hierarchical Interface-based Supervisory Control* ». Thèse de doctorat, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, 2003.

[35] R. J. LEDUC, B. A. BRANDIN, M. LAWFORD et W. M. WONHAM. « Hierarchical interface-based supervisory control – part I : Serial case ». *IEEE Transactions on Automatic Control*, 50(9) :1322–1335, 2005.

[36] R. J. LEDUC, P. DAI et R. SONG. « Synthesis method for hierarchical interface-based supervisory control ». *IEEE Transactions on Automatic Control*, 54(7) :1548–1560, 2009.

[37] R. J. LEDUC, M. LAWFORD et P. DAI. « Hierarchical interface-based supervisory control of a flexible manufacturing system ». *IEEE Transactions on Control Systems Technology*, 14(4) :654–668, 2006.

[38] R. J. LEDUC, M. LAWFORD et W. M. WONHAM. « Hierarchical interface-based supervisory control – part II : Parallel case ». *IEEE Transactions on Automatic Control*, 50(9) :1336–1348, 2005.

[39] C. MA et W. M. WONHAM. *Nonblocking Supervisory Control of State Tree Structures*, volume 317 de *Lecture Notes in Control and Information Sciences*. Springer-Verlag, Berlin, 2005.

[40] A. S. MINER et G. CIARDO. « Efficient reachability set generation and storage using decision diagrams ». Dans S. DONATELLI et H. C. M. KLEIJN, éditeurs, *Proc. of the 20th International Conference on Application and Theory of Petri Nets*, volume 1639 de *Lecture Notes in Computer Sciences*, pages 6–25, London, UK, 1999. Springer-Verlag.

[41] R. PAIGE et R. E. TARJAN. « Three partition refinement algorithms ». *SIAM Journal on Computing*, 16(6) :973–989, 1987.

[42] K. Q. PU. « Modeling and Control of Discrete-Event Systems with Hierarchical Abstraction ». Mémoire de maîtrise, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, 2000.

[43] P. J. G. RAMADGE et W. M. WONHAM. « Modular feedback logic for discrete event systems ». *SIAM Journal on Control and Optimization*, 25(5) :1202–1218, 1987.

[44] P. J. G. RAMADGE et W. M. WONHAM. « Modular supervisory control of discrete-event systems ». *Mathematics of Control, Signals and Systems*, 1(1) :13–30, 1988.

[45] P. J. G. RAMADGE et W. M. WONHAM. « The control of discrete event systems ». *Proceedings of the IEEE*, 77(1) :81–98, 1989.

[46] J. RICHARDSSON et M. FABIAN. « Automatic generation of PLC programs for control of flexible manufacturing cells ». Dans *Proc. IEEE Conference Emerging Technologies and Factory Automation ETFA 2003*, volume 2, pages 337–344, Lisbon, Portugal, 2003.

[47] K. RUDIE. « The integrated discrete-event systems tool ». Dans *Proc. of the 8th International Workshop on Discrete Event Systems*, pages 394–395, Ann Arbor, Michigan, 2006.

[48] B. SCHWARTZ. « State Aggregation of Controlled Discrete-Event Systems ». Mémoire de maîtrise, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, 1992.

[49] A. C. SHAW. *Real-Time Systems and Software*. John Wiley & Sons Inc., 2001.

[50] SIEMENS. « *S7-SCL V5.3 pour S7-300/400* ». Siemens AG, Nürnberg, Germany, 2005.

[51] SIEMENS. « *Manuel CPU 31xC et CPU 31x, Caractéristiques techniques* ». Siemens AG, Nürnberg, Germany, 2006.

[52] R. SONG. « Symbolic Hierarchical Interface-based Supervisory Control ». Mémoire de maîtrise, McMaster University, Hamilton, Canada, 2006.

[53] R. SU et W. M. WONHAM. « Supervisor reduction for discrete-event systems ». *Discrete Event Dynamic Systems*, 4(1) :31–53, 2004.

[54] B. WANG. « Top-down Design for RW Supervisory Control Theory ». Mémoire de maîtrise, Department of Electrical and Computer Engineering, University of Toronto, Toronto, 1995.

[55] K. C. WONG. « *Discrete-Event Control Architecture : an Algebraic Approach* ». Thèse de doctorat, University of Toronto, Toronto, Canada, 1994.

[56] K. C. WONG. « On the Complexity of Projections of Discrete-Event Systems ». Dans *Proc. of IEEE Workshop on Discrete Event Systems*, pages 201–206, Cagliari, Italy, 1998.

[57] K. C. WONG, J. G. THISTLE, R. P. MALHAMÉ et H.-H. HOANG. « Supervisory control of distributed systems : conflict resolution ». *Discrete Event Dynamic Systems : Theory and Applications*, 10(1) :131–186, 2000.

[58] K. C. WONG et W. M. WONHAM. « Hierarchical control of discrete-event systems ». *Discrete Event Dynamic Systems*, 6(3) :241–273, 1996.

[59] K. C. WONG et W. M. WONHAM. « Modular control and coordination of discrete-event systems ». *Discrete Event Dynamic Systems*, 8(3) :247–297, 1998.

[60] K. C. WONG et W. M. WONHAM. « On the computation of observers in discrete-event systems ». *Discrete Event Dynamic Systems*, 14(1) :55–107, 2004.

[61] W. M. WONHAM. « Towards an abstract internal model principle ». *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(11) :735–740, November 1976.

[62] W. M. WONHAM et E. S. ROGERS, Sr. « Notes on Control of Discrete-event Systems ». Lecture Notes ECE 1636F/1637S 2007-08, University of Toronto, Toronto, Canada, 2007.

[63] Z. H. ZHANG. « Smart TCT : An Efficient Algorithm for Supervisory Control Design ». Mémoire de maîtrise, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, 2001.

[64] Z. H. ZHANG et W. M. WONHAM. « STCT : An efficient algorithm for supervisory control design ». Dans B. CAILLAUD, P. DARONDEAU, L. LAVAGNO et X. XIE, éditeurs, *Synthesis and Control of Discrete Event Systems*, pages 77–100, Netherlands, 2002.

[65] H. ZHONG et W. M. WONHAM. « Hierarchical coordination ». Dans *Proc. of 5th IEEE International Symposium on Intelligent Control*, volume 1, pages 8–14, 1990.

[66] H. ZHONG et W. M. WONHAM. « On the consistency of hierarchical supervision in discrete-event systems ». *IEEE Transactions on Automatic Control*, 35(10) :1125–1134, 1990.