UNIVERSITÉ DE MONTRÉAL

Vers la construction automatique d'un module inconnu dans un système composé

Par

Jawad Drissi

Département d'Informatique et de Recherche Opérationnelle
Faculté des Arts et des Sciences

Thèse présentée à la Faculté des Études Supérieures
en vue de l'obtention du grade de
Philosophiæ Doctor (Ph. D.)
en Informatique

Mars 2000

© Jawad Drissi, 2000

QA
76
U54
2000
v. 035

Université de Montréal
Faculté des Études Supérieures

Cette thèse intitulée:

Vers la construction automatique d'un module
inconnu dans un système composé

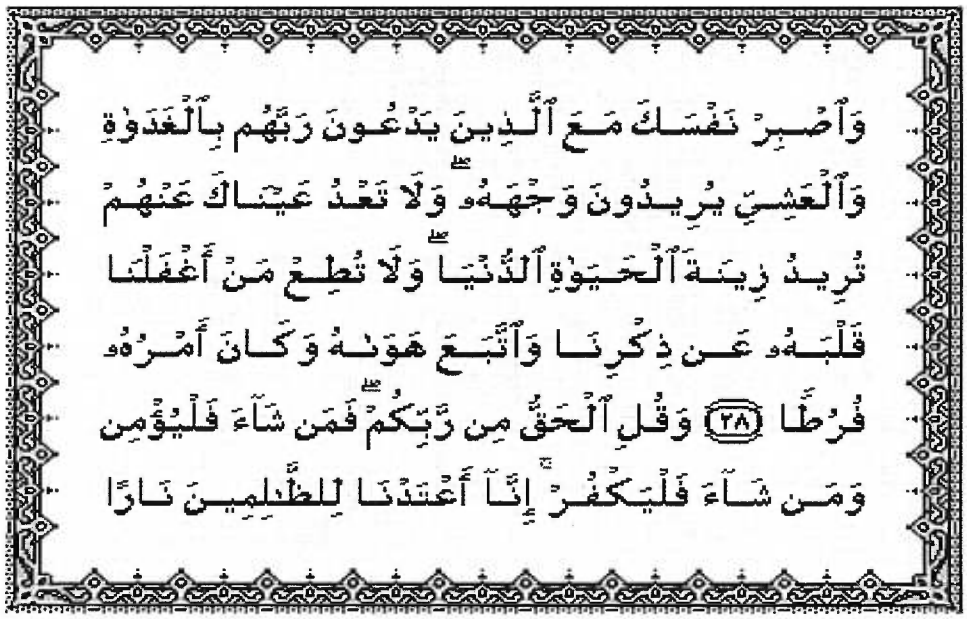
présentée par:

Jawad Drissi

a été évaluée par un jury composé des personnes suivantes:

Professeur Peter Kropf	Président du Jury
Professeur Gregor v. Bochmann	Directeur de recherche
Professeur Joseph Sifakis	Examinateur externe
Professeur El Mostapha. Aboulhamid	Membre du jury

Thèse acceptée le: 26 juillet 2000


وَٱصۡبِرۡ نَفۡسَكَ مَعَ ٱلَّذِينَ يَدۡعُونَ رَبَّهُم بِٱلۡغَدَوٰةِ
وَٱلۡعَشِيِّ يُرِيدُونَ وَجۡهَهُۥۖ وَلَا تَعۡدُ عَيۡنَاكَ عَنۡهُمۡ
تُرِيدُ زِينَةَ ٱلۡحَيَوٰةِ ٱلدُّنۡيَاۖ وَلَا تُطِعۡ مَنۡ أَغۡفَلۡنَا
قَلۡبَهُۥ عَن ذِكۡرِنَا وَٱتَّبَعَ هَوَىٰهُ وَكَانَ أَمۡرُهُۥ
فُرُطٗا ﴿٢٨﴾ وَقُلِ ٱلۡحَقُّ مِن رَّبِّكُمۡۖ فَمَن شَآءَ فَلۡيُؤۡمِن
وَمَن شَآءَ فَلۡيَكۡفُرۡۚ إِنَّآ أَعۡتَدۡنَا لِلظَّٰلِمِينَ نَارًا

Fais preuve de patience [en restant] avec ceux qui invoquent leur Seigneur matin et soir, désirant Sa Face. Et que tes yeux ne se détachent point d'eux, en cherchant (le faux) brillant de la vie sur terre. Et n'obéis pas à celui dont Nous avons rendu le cœur inattentif à Notre Rappel, qui poursuit sa passion et dont le comportement est outrancier. Et dis: ‹La vérité émane de votre Seigneur›. Quiconque le veut, qu'il croit, et quiconque le veut qu'il mécroie.

Coran, Sourate Al Kahf, versets 28-29.

SOMMAIRE

Cette thèse présente une solution générale pour la construction d'un composant à ajouter dans un système dans le cas où les autres composants ainsi que le système global désiré sont donnés. Le composant à construire peut d'un coté être considéré comme un sous système communicant avec les autres sous systèmes donnés afin que l'ensemble des sous systèmes fournisse le service défini par la spécification du système global désiré. Il peut d'un autre coté être considéré comme un contrôleur qui commande le comportement des sous systèmes donnés, en autorisant et interdisant leurs événements commandables de manière à satisfaire les contraintes définies par la spécification du système global désiré.

Dans notre travail, on s'est concentré principalement sur la résolution du problème de la construction de sous modules dans le cadre des systèmes qui communiquent par entrées et sorties, c'est à dire, par envoi et réceptions de messages. La communication entre deux sous systèmes est modélisée par une composition synchrone d'une entrée d'un sous système avec une sortie de l'autre. Dans ce type de communication, une émission d'un message ne peut jamais être refusée. C'est cette absence de refus qui caractérise un modèle de communication par entrées et sorties, à la différence du mode de communication par rendez-vous, comme en LOTOS, où la notion de refus est au contraire très importante.

Le problème de construction de sous modules est décrit formellement par l'équation $(C \| X) \mathfrak{R} A$, où C représente la spécification de la partie connue ou existante du système, appelée le contexte, A représente la spécification du système entier, $\|$ décrit le mode de regroupement des modules c'est à dire un opérateur de composition et $\mathfrak{R}$ décrit une relation de conformité entre systèmes. Dans cette formulation, l'ensemble des entrées du module à concevoir est défini de façon implicite. C'est la réunion des sorties internes du contexte et des actions reçues par le système à partir de l'environnement ne faisant pas partie des entrées du contexte. Nous avons jugé plus opportun de laisser le choix de cette ensemble au concepteur. Dans cette optique, nous avons opté pour une définition explicite de cet ensemble afin de permettre au module à concevoir d'observer certaines interactions du contexte avec l'environnement. Dans le cas extrême, le module à concevoir peut observer

toutes les actions présentes dans le système. Par contre, dans les travaux antérieurs nous avons le cas de l'observabilité minimale pour le module à concevoir. Le problème de construction de sous modules consiste dans ce cas à résoudre l'équation $(C||X)\Re A$ sous la contrainte $I_X=In$ où In est l'ensemble des entrées requises pour le module à concevoir. Au lieu de développer pour chaque relation de conformité un algorithme indépendant pour résoudre le problème de construction de sous modules, nous avons défini une relation de conformité générique, l'implantation conforme. Cette relation est inspirée de la relation sous-type présente dans les langages orienté-objet. C'est une relation de conformité générique dans le sens où chacune des relations de conformité bien connues trace équivalence, réduction et quasi-équivalence peut être considérée comme un cas particulier. Pour définir cette relation, nous avons utilisé un nouveau type d'automates que nous avons appelé automate à entrées et sorties avec traces complètes optionnelles. Ces automates permettent de prendre en compte la notion de tache complète, qui impose qu'après une trace donnée le système doit progresser afin de produire certaines sorties, ainsi que la notion de choix, qui permet à une implantation conforme d'avoir au moins un comportement parmi un ensemble de comportements décrient par la spécification.

Par la suite, nous avons proposé une méthode pour la résolution du problème de construction de sous modules dans le cadre décrit précédemment. Un algorithme a été développé pour l'obtention de la solution la plus générale lorsque la relation de conformité utilisée est la relation implantation conforme. De plus, nous avons donné une caractérisation de l'ensemble des solutions possibles de l'équation. Par la suite, une implantation de l'algorithme a été réalisée dans le cadre du développement d'un outil pour la construction de sous modules. Un dernier travail a porté sur l'extension de l'algorithme précédant au cas des systèmes temps-réel.

TABLE DES MATIÈRES

LISTE DES FIGURES

REMERCIEMENTS

Mes remerciements vont en premier lieu au Professeur Gregor v. Bochmann, qui a accepté d'encadrer mon travail de recherche. Sa grande expérience, la rigueur de sa démarche scientifique, sa disponibilité et son humanisme m'ont aidé constamment pendant les travaux de recherche et la rédaction de cette thèse.

Je tiens à remercier tous les membres du groupe de téléinformatique et protocoles de communication de l'Université de Montréal qui, à un titre ou un autre, ont contribués à la réalisation de cette recherche.

Mes remerciements vont également à mes Parents, ma soeur et mes frères pour leur soutien continu durant toute la période de mes études.

Je suis très reconnaissant à ma femme qui m'a aidé avec beaucoup de patience et de compréhension à mener à terme mon travail de recherche. Je remercie Dieu de m'avoir donné un fils dont la présence, à elle seule, est une grande source de joie.

Finalement, je remercie le "NSERC strategic grant STRGP200" pour le soutien financier dont j'ai bénéficié durant toute la durée de cette recherche.

CHAPITRE 1

INTRODUCTION

L'évolution technologique a conduit au développement de systèmes informatiques complexes, dont l'impact socio-économique est devenu très fort, dans la mesure où ils occupent des places de plus en plus stratégiques au sein des organisations. De tels systèmes intègrent de nombreux composants logiciels et matériels et interagissent avec des environnements complexes. Ces systèmes sont devenus critiques tant par les conséquences de leur utilisation que par la complexité de leur développement et de leur évolution. Une classe importante des systèmes critiques est celle des systèmes réactifs, systèmes interagissant de façon continue avec un environnement.

Produire des systèmes réactifs répondant à des exigences de qualité données, à des coûts et dans des délais raisonnables, est un enjeu économique majeur et également un défi scientifique et technologique important. Pour répondre à ce défi il faut disposer entre autres d'outils et de méthodes pour améliorer l'efficacité du développement.

1.1 Motivations

Une société installée à Montréal est propriétaire d'un camion. Un de ses clients désire transporter des marchandises de Montréal à Ottawa. Afin de rendre le service désiré par le client, la société devra embaucher une personne sachant conduire un camion et connaissant le trajet reliant Montréal à Ottawa. Cet exemple illustre de manière intuitive le problème auquel on s'intéresse dans cette thèse. Le besoin du client spécifie le service requis. Le camion possédé par la société décrit l'existant. Le complément nécessaire pour la réalisation du service est dans ce cas le chauffeur.

Ce genre de problème se rencontre, lors de la conception hiérarchique de systèmes

complexes, dans la synthèse de contrôleurs ainsi que dans la réutilisation de composantes. C'est le problème de construction de sous-modules, aussi appelé problème de factorisation ou problème de résolution d'équation. Le problème de construction de sous-modules (Figure 1.1) consiste dans la construction d'un sous-module X d'un système lorsque les spécifications du système entier et de tous ses sous-modules à l'exception de celle du sous-module X sont données.

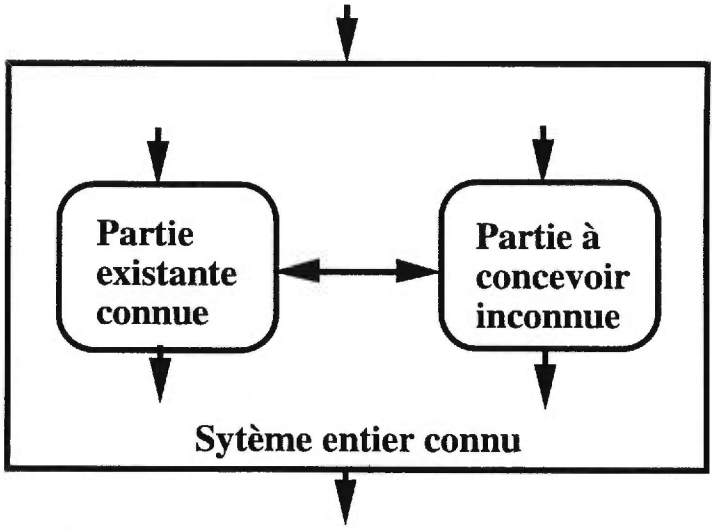


Figure 1.1 : Le problème de construction de sous-modules

Dans les réseaux informatiques des protocoles sont définis. Un protocole est un ensemble de règles régissant les interactions entres deux entités communicantes [Bochmann 90]. Il est souvent décrit à deux niveaux d'abstraction, la spécification du service et la spécification du protocole. La spécification du service décrit le comportement à partir d'un point de vue utilisateur. Elle définit un ensemble de primitives de services grâce auxquelles l'utilisateur pourra accéder aux services fournis par le système. Lors de l'interconnexion de deux réseaux utilisant des protocoles incompatibles, il est nécessaire d'utiliser une passerelle [Green 86]. La passerelle devra réaliser les transformations nécessaires pour un message ou un ensemble de messages provenant d'une entité d'un protocole avant de l'envoyer vers une entité de l'autre protocole tout en préservant la sémantique qui lui est associée. La construction de la passerelle peut être vue comme un cas particulier du problème de construction de sous-modules. Dans ce cas, les deux réseaux individuels décrivent l'existant, le réseau global obtenu après l'interconnexion des deux réseaux décrit le système entier désiré et la passerelle sera la partie manquante du système qu'il faudra concevoir (Figure 1.2). On peut trouver des exemples de telles applications dans [Tao 95].

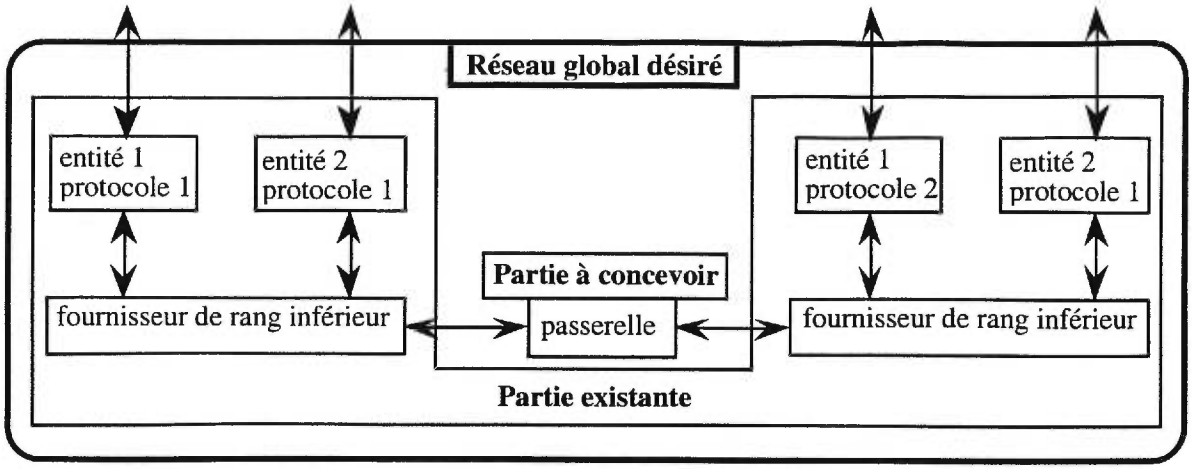


Figure 1.2 : La construction d'une passerelle

Dans les systèmes à événements discrets [Ramadge 89], un ensemble de composantes coopèrent pour la réalisation d'une tâche donnée. De tels systèmes servent par exemple pour décrire les systèmes de production englobant des robots et des chaînes de montage. La principale caractéristique de ces systèmes est d'être constitués d'activités concurrentes communiquant par l'échange de signaux discrets. Pour maintenir un état de fonctionnement cohérent du système, il faut concevoir une composante qui assurera le contrôle et la coordination des différentes composantes du système. La construction du contrôleur peut aussi être vue comme un cas particulier du problème de construction de sous-modules.

1.2 Formalisation du problème

Le problème de construction de sous-modules peut être décrit formellement par l'équation $(C||X)\mathfrak{R}A$, où C représente la spécification de la partie connu ou existante du système, appelée le contexte, A représente la spécification du système entier, $||$ décrit le mode de regroupement des modules c'est à dire un opérateur de composition et $\mathfrak{R}$ décrit une relation de conformité entre systèmes.

Cette équation est en réalité à multiples facettes. Elle nécessite dans un premier temps le choix d'un formalisme pour la description des spécifications du système entier et de la partie existante du système. Ce formalisme servira aussi à décrire la solution recherchée. Une fois ce formalisme choisi, il faudra définir l'opérateur de composition qui nous permettra de décrire le système obtenu suite à la coopération de la partie existante du système avec la partie à concevoir. Enfin, la relation de conformité devra être précisée. Par la suite, il faudra choisir si l'on désire obtenir au moins une solution ou si l'on désire obtenir l'ensemble des solutions

pour l'équation afin d'avoir la possibilité de choisir une solution particulière selon certains critères d'optimalité (par exemple nombre minimal d'états).

Plusieurs travaux se sont intéressés au problème de construction de sous-modules. Dans le travail [Merlin 83], les spécifications sont décrites sous forme de séquences d'exécution, et l'équivalence de traces est utilisée comme critère de conformité. Dans [Shields 89], l'auteur utilise CCS (Calculus of Communicating Systems) de Milner pour modéliser le même problème en utilisant l'équivalence observationnelle comme relation de conformité. Le travail s'intéresse aux conditions nécessaires et suffisantes pour l'existence d'une solution. Le travail de Qin et Lewis [Qin 1991] étend celui de Shields en proposant un algorithme pour l'obtention de la solution la plus générale (une solution qui simule toutes les solutions possibles). Parrow [Parrow 89] présente une méthode basée sur des transformations successives d'équations en des équations plus simples en parallèle afin de générer une solution. La méthode est semi-automatique. Dans [Tao 96], l'auteur présente une méthode où les événements observable ne sont pas nécessairement contrôlables. Dans [Watanabe 93a], les auteurs présentent une méthode pour la résolution du problème de construction de sous-modules lorsque les spécifications sont données sous la forme de machines de Mealy communiquant de façon synchrone.

Dans le domaine de la synthèse de contrôleur, on peut citer les travaux suivants. Ramadge et Wonham [Ramadge 87][Ramadge 89] ont développé un cadre théorique basé sur les automates pour la définition et la résolution de problèmes de synthèse de contrôleurs pour des systèmes à événements discrets. Thistle et Wonham [Thistle 94][Thistle 95] ont proposé une méthode pour caractériser le sous ensemble des états d'un automate de Rabin à partir desquels l'automate peut être contrôlé pour ne produire que des comportements (traces) qui satisfont les conditions d'acceptation associées à l'automate. Maler, Pnueli et Sifakis [Maler 95] présentent un algorithme pour la synthèse automatique de contrôleurs lorsque les spécifications sont données sous la forme d'automates temporisés [Alur 94]. Dans [Aziz 95], les auteurs présentent une méthode pour la synthèse de contrôleurs lorsque les spécifications sont données sous la forme de machines de Mealy communiquant de façon synchrone.

1.3 Contributions

Dans cette thèse, nous nous sommes intéressé dans un premier temps au problème de construction de sous-modules dans le modèle des machines de Mealy déterministes complètement spécifiées [Drissi 98b]. La relation de conformité utilisée est l'équivalence de traces. Nous avons présenté un algorithme pour la construction d'une machine de Mealy qui

contient seulement les traces permises pour le module à concevoir. Cette machine est appelée la solution générique potentielle. Toute solution de l'équation est une réduction de la solution générique potentielle, mais certaines réductions de cette dernière machine ne sont pas des solutions car leurs compositions avec le contexte ne peuvent être modélisées par une machine de Mealy à cause de cycles étiquetés seulement par des actions internes. À la différence du travail de Watanabe et al. [Watanabe 93a], nous considérons que la communication entre les composantes est asynchrone. De plus, nous supposons que le module à concevoir peut interagir avec le contexte et/ou l'environnement.

La solution générique potentielle est obtenue sous la forme d'une machine de Mealy non déterministe. Dans le cas où toutes ces réductions sont des solutions, il peut être intéressant d'obtenir une solution dont le nombre d'états est minimal. Nous décrivons dans le travail [Drissi 98a] une méthode pour la génération d'une implantation minimale à partir d'une spécification non déterministe.

Une première approche pour la généralisation du travail [Drissi 98b] consistait à procéder étape par étape. La première étape étant la résolution du problème de construction de sous-modules dans le modèle des machines de Mealy non déterministes complètement spécifiées. Le travail de Petrenko et Yevtushenko [Petrenko 98] se situe dans cette approche. Il présente de plus une technique pour traiter les cycles étiquetés seulement par des actions internes. Le cas des machines de Mealy partiellement spécifiées restait ouvert dans ce travail.

De notre coté nous avons privilégié une autre approche. Dans un premier temps, nous avons généralisé le modèle en utilisant celui des automates à entrées et sorties. Afin de pouvoir considérer une machine de Mealy comme un cas particulier de notre modèle nous avons opté pour des automates à entrées et sorties partiellement spécifiés. Un automate à entrées et sorties observe toutes les actions présentes dans son alphabet mais ne contrôle que les actions produites par lui, c'est-à-dire ses sorties. Dans le travail sur les machines de Mealy, l'ensemble des entrées du module à concevoir est défini de façon implicite. C'est la réunion des sorties internes du contexte et des actions reçues par le système à partir de l'environnement ne faisant pas partie des entrées du contexte. Dans un deuxième temps, nous avons opté pour une définition explicite de cet ensemble afin de permettre au module à concevoir d'observer certaines interaction du contexte avec l'environnement. Le problème de construction de sous-modules consiste dans ce cas à résoudre l'équation $(C||X)\Re A$ sous la contrainte $I_X=In$ où In est un ensemble d'actions vérifiant $(I_A \setminus I_C) \cup (O_C \setminus O_A) \subseteq In \subseteq I_A \cup O_C$. Dans le cas où $In=I_A \cup O_C$, le module à concevoir observe toutes les actions présentes dans le

système. Par contre si $In=(I_A \setminus I_C) \cup (O_C \setminus O_A)$, nous avons l'observabilité minimale pour le module à concevoir. C'est ce dernier cas qui était considéré pour les machines de Mealy. Il faut noter à ce niveau que l'existence d'une solution dépend du degré d'observation alloué au module à concevoir. Au lieu de résoudre le problème de construction de sous-modules pour les relations de conformité bien connues telle l'équivalence de trace, la réduction et la quasi-équivalence, nous avons dans un troisième temps défini deux relations de conformité. La première relation de conformité, la réalisation sécuritaire, capture ce qui est commun aux trois relations de conformité précédentes. La deuxième relation de conformité, l'implantation conforme, est inspirée de la relation sous-type présente dans les langages orienté-objet. C'est une relation de conformité générique dans le sens où chacune des relations de conformité trace équivalence, réduction et quasi-équivalence peut être considérée comme un cas particulier de la relation implantation conforme. Pour définir cette relation, nous avons utilisé un nouveau type d'automates que nous avons appelé automates à entrées et sorties avec traces complètes optionnelles. Ces automates permettent de prendre en compte la notion de tâche complète, qui impose qu'après une trace donnée le système doit progresser afin de produire certaines sorties, ainsi que la notion de choix, qui permet à une implantation conforme d'avoir au moins un comportement parmi un ensemble de comportements décrits par la spécification.

Par la suite, nous avons proposé deux algorithmes pour la construction de sous-modules et prouvé qu'ils sont corrects [Drissi 99a]. Le premier (respectivement deuxième) algorithme permet de générer la solution générique de l'équation $(C \| X) \Re A$ sous la contrainte $I_X=In$ lorsque la relation de conformité utilisée est la réalisation sécuritaire (respectivement implantation conforme). De plus, dans chacun des cas nous avons caractérisé l'ensemble des solutions possibles de l'équation.

Le langage de programmation orienté-objet Java a été utilisé pour l'implantation des deux algorithmes précédents pour développer un outil pour la construction de sous-modules [Drissi 99b]. Le but principal de l'outil est la génération de la solution générique pour le problème de la construction de sous-modules dans le cas où les spécifications sont données sous la forme de machines de Mealy ou bien sous la forme d'automates à entrées et sorties et ceci pour chacune des relations de conformité mentionnées dans le paragraphe précédant.

Dans un dernier travail, nous nous sommes intéressé à la résolution de l'équation $(C \| X) \Re A$ sous la contrainte $I_X=In$ lorsque les spécifications sont données sous la forme d'automates à entrées et sorties temporisés dans le cas où la relation de conformité est la

réalisation sécuritaire [Drissi 99c]. Dans le travail [Maler 95], les auteurs supposent que le contrôleur à dériver observe exactement l'état du système à contrôler à tout moment. Dans notre travail nous éliminons cette hypothèse et nous considérons le cas où le module à concevoir doit fonctionner avec une certaine incertitude sur l'état exact du contexte.

1.4 Plan de la thèse

Dans le chapitre 2, nous présentons les définitions des modèles des systèmes de transitions qui seront utilisés dans cette thèse ainsi que leurs propriétés de base. Les différents modèles sont : le modèle des systèmes de transitions étiquetées, le modèle des automates à entrées et sorties et le modèle des machines de Mealy.

Dans le chapitre 3, nous définissons pour chacun des modèles présentés dans le chapitre 2 un opérateur de composition, des relations de conformité ainsi que certaines transformations permettant de passer d'un système à transitions à un autre sous une forme jugée plus adéquate.

Dans le chapitre 4, nous passons en revue les différents travaux qui se sont intéressés au problème de construction de sous-modules dans le modèle des systèmes de transitions étiquetées.

Dans le chapitre 5, nous présentons une approche pour la résolution du problème de construction de sous-modules. Dans cette approche, le modèle des machines de Mealy déterministes complètement spécifiées est utilisé comme formalisme pour décrire les spécifications, et la relation de conformité choisie est l'équivalence de traces.

Dans le chapitre 6, nous proposons une solution pour la détermination d'une implantation déterministe minimale dont les traces sont incluses dans celles d'une machine de Mealy non déterministe.

Dans le chapitre 7, nous généralisons le travail du chapitre 5. Nous présentons une approche pour la résolution du problème de construction de sous-modules lorsque le modèle des automates à entrées et sorties est utilisé comme formalisme pour décrire les spécifications. Cette approche permet l'utilisation de différentes relations de conformité de manière uniforme.

Dans le chapitre 8, on s'intéresse aux automates à entrées et sorties temporisés. Nous présentons une approche pour la résolution du problème de construction de sous-modules

dans ce modèle en considérant pour relation de conformité la réalisation sécuritaire.

Dans le chapitre 9, nous présentons l'outil développé en Java pour la construction de sous-modules. Cet outil implante les algorithmes du chapitre 5 et ceux du chapitre 7. Le but principal de l'outil est la génération de la solution générique pour le problème de la construction de sous-modules dans le cas où les spécifications sont données sous la forme de machines de Mealy ou bien sous la forme d'automates à entrées et sorties et ceci pour une variété de relations de conformité.

Finalement, au chapitre 10, nous présentons les principales conclusions de la présente recherche, ainsi que quelques directions possibles pour des recherches futures.

CHAPITRE 2

LES MODÈLES DES SYSTÈMES À TRANSITIONS

L'idée d'un système qui reçoit des signaux discrets (les entrées) de l'extérieur et qui produit d'autres signaux (les sorties) qui agissent sur l'extérieur, est très générale. Elle est appliquée par exemple avec plus ou moins de succès pour modéliser le fonctionnement de circuits digitaux électroniques, la génération ou la reconnaissance de langages formels et le comportement de systèmes biologiques. C'est donc naturel qu'on ait développé une théorie mathématique pour exprimer de façon abstraite et unifiée ce qui est fondamental dans les systèmes discrets du traitement de l'information. Les systèmes de transitions finis sont un des formalismes utilisés pour décrire de tels systèmes. Bien que mathématiquement très simple, ce formalisme permet de modéliser, avec une précision souvent suffisante, la plupart des aspects des systèmes de processus, et joue un rôle important dans la définition de leur sémantique. La plupart des travaux sur la sémantique des processus dits parallèles ou communicants reposent sur la notion d'automate. Un automate fini, constitué d'états et de transitions étiquetées entre ces états, permet de décrire un système dont l'état évolue au cours du temps. À partir d'une représentation d'un système sous forme d'automate, il est possible d'observer et de vérifier certaines propriétés de ce système, telles que la présence de situation de blocage.

Dans ce qui suit, nous présentons les définitions des modèles des systèmes de transitions qui seront utilisés dans cette thèse ainsi que leurs propriétés. Nous nous limitons à des systèmes dont les ensembles d'états et d'interactions sont finis. Cette restriction ne nous pénalise pas, car les systèmes réels peuvent toujours être modélisés par des automates finis.

2.1 Systèmes de transitions étiquetées

Le modèle des systèmes de transitions étiquetées est utilisé pour décrire la sémantique

d'un certain nombre de langages de spécification, comme CCS [Milner 80], CSP [Hoare 85] ou LOTOS [IS8807 89], car il représente bien les processus communicants décrits dans ces langages. Il est également utilisé dans des travaux sur le test [Brinksma 88][Brinksma 89][Yao 93][Tan 95].

Définition 2.1 : Un système de transitions étiquetées finis (LTS : Labeled Transition System) est défini par la donnée d'un quadruplé (S, L, T, s_0) où :

- S est un ensemble fini non vides d'états;
- L est un ensemble fini non vide d'actions observables;
- $T \subseteq S \times (L \cup \{\tau\}) \times S$ est la relation de transition où $\tau \notin L$ est une action interne non observable;
- s_0 est l'état initial du système de transitions étiquetées.

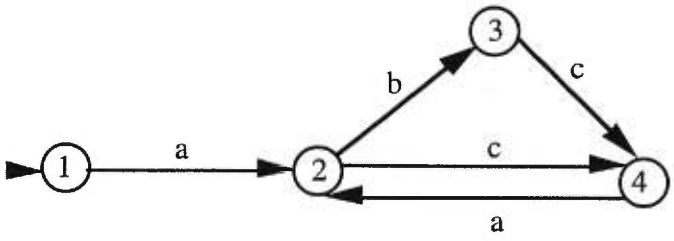


Figure 2.1 : Représentation graphique d'un système de transitions étiquetées

La figure 2.1 illustre la représentation graphique du système de transitions étiquetées $E=(\{1, 2, 3, 4\}, \{a, b, c\}, \{(1, a, 2), (2, b, 3), (2, c, 4), (3, c, 4), (4, a, 2)\}, 1\}$.

Dans la suite, pour un LTS $E=(S, L, T, s_0)$ et $\sigma = \mu_1 \mu_2 \ldots \mu_n \in L^n$, nous utiliserons les notations suivantes:

- (s_1, σ, s_{n+1}) si $(\exists (s_i)_{1 \le i \le n+1} \in S^{n+1})(\forall i, 2 \le i \le n+1)((s_{i-1}, \mu_{i-1}, s_i) \in T)$,
- (E, σ, s_n) si (s_0, σ, s_n),
- (s_1, ε, s_2) si $s_1 = s_2$ ou $(\exists n \ge 1)$ (s_1, τ^n, s_2),
- $(s_1, \vec{\mu}, s_2)$ si $(\exists s_3, s_4 \in S)$ $((s_1, \varepsilon, s_3) \wedge (s_3, \mu, s_4) \wedge (s_4, \varepsilon, s_2))$,
- $(s_1, \vec{\sigma}, s_{n+1})$ si $(\exists (s_i)_{1 \le i \le n+1} \in S^{n+1})(\forall i, 2 \le i \le n+1)(s_{i-1}, \overrightarrow{\mu_{i-1}}, s_i)$,
- *act_vis_possible* $(s)=\{\mu \in L \mid \exists s' \in S\ (s, \vec{\mu}, s')\}$.

Une séquence d'actions observables σ est dite une trace observable à partir de l'état s s'il existe un état s' tel que $(s, \vec{\sigma}, s')$. L'ensemble des traces observables à partir de l'état s est noté $Tr_E(s)$. L'ensemble des traces observables à partir de l'état initial s_0 représente l'ensemble des traces observables du système à transitions étiquetées E, on le note Tr_E.

Un système à transitions étiquetées est dit non déterministe lorsqu'ils existent au moins un état s et une séquence d'actions observables σ tels que $(s, \vec{\sigma}, s_1)$, $(s, \vec{\sigma}, s_2)$ et $s_1 \neq s_2$.

Un état s' est accessible à partir d'un état s s'il existe une trace σ telle que $(s, \vec{\sigma}, s')$. Si $s=s_0$ alors s' est dit accessible.

La composante connexe contenant l'état initial, notée $CC(E)$, est le système à transitions étiquetées $CC(E)=(S_C, L_C, T_C, s_{0C})$ tel que $S_C=\{s \in S \mid s \text{ accessible}\}$, $L_C=L$, $T_C=\{(s, u, s') \in T \mid s \in S_C\}$ et $s_{0C}=s_0$. Si $E=CC(E)$ alors I est initialement connecté.

2.2 Les automates à entrées et sorties

Une variante des systèmes à transitions étiquetées est obtenue en subdivisant l'ensemble des actions en deux sous ensembles, les entrées, c'est à dire les réceptions, et les sorties, c'est à dire les émissions. Ceci permet de distinguer les actions contrôlées par le système - les sorties- de celles qui sont contrôlées par son environnement -les entrées. Le modèle obtenu est celui des automates à entrées et sorties [Lynch 88]. Ce modèle a été utilisé pour modéliser des composantes autonomes dans les systèmes concurrents distribués.

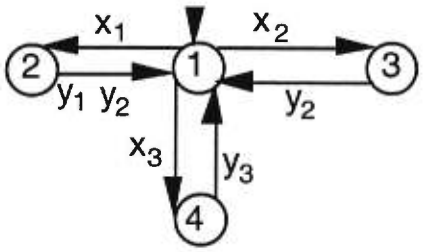


Figure 2.2 : Représentation graphique d'un automate à entrées et sorties

Définition 2.2 : Un automate à entrées et sorties (IOA : Input-Output Automaton) est défini par la donnée d'un quintuplé $(S_A, I_A, O_A, T_A, s_{0A})$ où :

- S_A est un ensemble fini non vides d'états;
- I_A est un ensemble fini non vide d'entrées;
- O_A est un ensemble fini non vide de sorties avec $I_A \cap O_A=\emptyset$;
- $T_A \subseteq S_A \times ((I_A \cup O_A) \cup \{\tau\}) \times S_A$ est la relation de transition où τ est une action interne non observable. Un élément $(s, u, s') \in T_A$ est noté $s\text{-}u\rightarrow s'$. Chaque élément de T_A est une transition, entre un état de départ et un état d'arrivée. Cette transition est soit associée à une action observable (émission ou réception d'une interaction), soit

associée à une action interne étiquetée par τ.

- s_{0A} est l'état initial de l'automate.

La Figure 2.2 illustre la représentation graphique de l'automate à entrées et sorties $A=(\{1, 2, 3, 4\}, \{x_1, x_2, x_3\}, \{y_1, y_2, y_3\}, \{(1, x_1, 2), (1, x_2, 3), (1, x_3, 4), (2, y_1, 1), (2, y_2, 1), (3, y_2, 1), (4, y_3, 1)\}, 1\}$.

Pour un automate à entrées et sorties $A=(S_A, I_A, O_A, T_A, s_{0A})$, si l'on occulte la distinction entre les entrées et les sorties, on obtient le système à transitions étiquetées correspondant E en regroupant les ensembles I_A et O_A en un seul ensemble d'actions observables. Ce système à transitions étiquetées E est donc défini par $(S_A, I_A \cup O_A, T_A, s_{0A})$.

Pour chaque état $s \in S_A$, on note :

- $inp(s)=\{x \in I_A \mid \exists s' \in S_A \ s\text{-}x\rightarrow s'\}$,
- $out(s)=\{y \in O_A \mid \exists s' \in S_A \ s\text{-}y\rightarrow s'\}$,
- $entering(s)=\{u \in (I_A \cup O_A) \mid \exists s' \in S_A \ s'\text{-}u\rightarrow s\}$,
- $leaving(s)=inp(s) \cup out(s)$.

Si pour chaque $s \in S_A$ et tout $x \in I_A$ il existe $s' \in S_A$ telle que $s\text{-}x\rightarrow s'$, alors A est complètement spécifié (input-enabled); sinon A est partiellement spécifié.

La forme complétée d'un automate à entrées et sorties A, notée $Ief(A)$, est l'automate à entrées et sorties obtenu par l'ajout à chaque état s de A de transitions étiquetées par les éléments de $(I_A \backslash inp(s))$ menant à l'état spécial *Fail*, c'est à dire,

$$Ief(A)=(S_A, I_A, O_A, T_A \cup (\cup_{s \in S_A}(\{s\} \times (I_A \backslash inp(s)) \times \{Fail\})), s_{0A}).$$

Un automate à entrées et sorties A est non déterministe si ils existent $s\text{-}u\rightarrow s'$, $s\text{-}u\rightarrow s''$ et $s' \neq s''$ pour au moins un état s et une action u ou si l'action interne τ est présente, sinon A est déterministe.

Une trace d'un automate à entrées et sorties A à partir d'un état s est la donnée d'une séquence d'actions observables $\sigma \in (I_A \cup O_A)^*$ telle que $(\exists s_n \in S)$ $(s, \vec{\sigma}, s_n)$. L'ensemble des traces de A à partir de s est noté $Tr_A(s)$ et on le note Tr_A dans le cas où $s=s_{0A}$. La notion de trace permet de faire abstraction des actions internes.

Un état s' d'un automate à entrées et sorties A est accessible à partir d'un état s s'il existe $\sigma \in Tr_A(s)$ telle que $(s, \vec{\sigma}, s')$. Si s est l'état initial alors s' est dit accessible.

Dans un automate à entrées et sorties A déterministe, un état s et une séquence $\sigma \in Tr_A(s)$ déterminent un unique état final après l'exécution de la trace σ, on le note s_σ. L'ensemble $S(s, \sigma)=\{s' \mid \exists \sigma'$ préfixe de σ tel que $s'=s_{\sigma'}\}$ représente les états accessibles à partir de l'état s par un préfixe de σ.

Pour une séquence $\sigma \in \Sigma^*$ et un sous ensemble Σ' de Σ, la projection de σ sur Σ', notée $Pr_{\Sigma'}(\sigma)$, est obtenu en éliminant de σ les symboles n'appartenant pas à Σ'. On note $Pr_A(\sigma)$, la projection de σ sur l'alphabet $(I_A \cup O_A)$ de l'automate à entrées et sorties A. Pour un ensemble de traces Y, on note $Pr_{\Sigma'}(Y)$, l'ensemble contenant les projections sur Σ' des éléments de Y. Pour un ensemble X contenant des ensembles de traces, on note $Pr_{\Sigma'}(X)$, l'ensemble $\{Pr_{\Sigma'}(Y) \mid Y \in X\}$.

La composante connexe contenant l'état initial, notée $CC(A)$, est l'automate à entrées et sorties $CC(A)=(S_C, I_C, O_C, T_C, s_{oC})$ tel que $S_C=\{s \in S_A \mid s$ est accessible$\}$, $I_C=I_A$, $O_C=O_A$, $T_C=\{(s, u, s') \in T_A \mid s \in S_C\}$ et $s_{oC}=s_{oA}$. Si $A=CC(A)$ alors A est initialement connecté.

2.3 Modèle des machines de Mealy

Le modèle, qui est le plus largement utilisé, est celui des machines de Mealy. Historiquement, ces automates ont été connus comme modèles pour étudier les propriétés des circuits électroniques où les sorties sont synchrones avec les entrées [Gill 62][Kohavi 78] [Starke 72]. Depuis les années 70 plusieurs travaux sur le test se basent sur ce modèle [Chow 78][Ural 87][Sidhu 89][Fujiwara 91]. Plus récemment, ce modèle a été utilisés dans le domaine de test de systèmes Orienté-Objet [Turner 92][Hoffman 93]. Ce modèle est aussi utilisé pour décrire la sémantique des langages de spécification SDL [Belina 89] et Estelle [ISO9074 89].

Définition 2.3 : Une machine de Mealy (FSM : Finite State Machine), est définie par la donnée d'un quintuplé $A=(S, X, Y, h, s_o)$ où :

- S est un ensemble fini non vide d'états;
- X un ensemble fini non vide de symboles d'entrée;
- Y un ensemble fini non vide de symboles de sorties;
- h: $D_A \subseteq S \times X \rightarrow \mathbb{P}(S \times Y) \setminus \emptyset$ est la fonction de comportement, où $\mathbb{P}(S \times Y)$ est l'ensemble des parties de $S \times Y$;
- s_o est l'état initial.

La figure 2.3 illustre la représentation graphique de la machine de Mealy $A=(\{1, 2\}, \{x_1, x_2\}, \{y_1, y_2\}, h, 1\}$ avec $h(1, x_1)=\{(1, y_1)\}$, $h(1, x_2)=\{(2, y_2)\}$, $h(2, x_1)=\{(2, y_2)\}$ et $h(2, x_2)=(1, y_1)$.

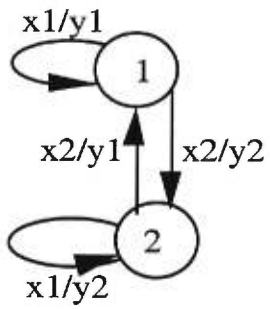


Figure 2.3 : Représentation graphique d'une machine de Mealy

Étant donné un état s et une entrée x, si $(p, y)\in h(s, x)$ alors il y a une transition de s vers p avec une entrée x et une sortie y. Une telle transition est notée s-x/y ->p.

La machine de Mealy A est dite observable, si on a : $|\{s'|\ (s',y)\in h(s,x)\}|\leq 1$ pour tout $(s, x)\in D_A$ et tout $y\in Y$ [Starke 72]. Ceci signifie que dans une machine observable, un état s et une entrée/sortie x/y tels que $(s, x)\in D_A$, déterminent un seul prochain état.

La machine est dite complètement spécifiée si $D_A=S\times X$, sinon elle est dite partiellement spécifiée.

La machine A est dite déterministe lorsqu'on a $|h(s, x)|=1$ pour tout $(s, x)\in D_A$, sinon elle est dite non déterministe. Dans une machine déterministe, on remplace la fonction de comportement h par deux fonctions: la fonction prochain état $\delta : D_A \rightarrow S$ et la fonction de sortie $\lambda : D_A \rightarrow Y$.

Une machine de Mealy $B=(S', X, Y, h', s_0)$ est appelée une sous-machine de A si :

$$S'\subseteq S \text{ et } h'(s, x)\subseteq h(s, x) \text{ pour tout } (s, x)\in S'\times X.$$

Si la sous-machine est déterministe, elle est dite une D-sous-machine de A.

Pour une machine de Mealy, on prolonge la fonction de comportement h sur $\bigcup_{s\in S} (\{s\}\times A_s^*)$, où A_s^* est l'ensemble de toutes les séquences d'entrées acceptées par l'état s de A, et on garde le même nom pour le prolongement, c'est à dire, $h: \bigcup_{s\in S} (\{s\}\times A_s^*) \rightarrow \mathbb{P}(S\times Y^*)$. Posons que $h(s, \varepsilon) = (s, \varepsilon)$ pour tout $s\in S$, et supposons que $h(s, \beta)$ est définie,

alors :

$$h(s, \beta x) = \{(s', \gamma y) \mid \exists\, s'' \in S\ [(s'', \gamma) \in h(s, \beta)\ \&\ (s', y) \in h(s'', x)]\}.$$

La première projection de h, notée h^1, est la fonction prochain état; alors que la seconde projection de h, notée h^2, est la fonction de sortie de A, c'est à dire,

$$h^1(s, \alpha) = \{\, s' \mid \exists\, \gamma \in Y^*\ ((s', \gamma) \in h(s, \alpha))\},$$
$$h^2(s, \alpha) = \{\, \gamma \mid \exists\, s' \in S\ ((s', \gamma) \in h(s, \alpha))\}.$$

Une séquence $\sigma = \mu_1\mu_2 \ldots \mu_{2n}$ telle que $\alpha = \mu_1\mu_3 \ldots \mu_{2n-1} \in X^n$ et $\beta = \mu_2\mu_4 \ldots \mu_{2n} \in Y^n$ est une trace à partir d'un état s d'une machine de Mealy $A=(S, X, Y, h, s_0)$ si lorsque on applique la séquence d'entrée α à partir de l'état s la machine A génère la séquence de sortie β. La séquence σ est aussi notée $\alpha \ltimes \beta$. L'ensemble des traces à partir d'un état s d'une machine de Mealy A est noté $Tr_A(s)$. L'ensemble des traces à partir de l'état initial s_0 représente l'ensemble des traces d'une machine de Mealy A, on le note Tr_A.

Considérons la classe des automates à entrées et sorties $A=(S_A, I_A, O_A, T_A, s_{0A})$ tels que A ne contient pas l'action invisible τ et l'ensemble des états S_A vérifie les propriétés suivantes :

i - S_A est égale à la réunion de deux sous ensembles disjoints S_{in} et S_{out},

ii - s_{0A} appartient à S_{in},

iii - pour chaque état s dans S_{in} on a $out(s)=\emptyset$ et $entering(s) \subseteq O_A$,

iv - pour chaque état s dans S_{out} on a $inp(s)=\emptyset$, $out(s)\neq\emptyset$ et $entering(s) \subseteq I_A$.

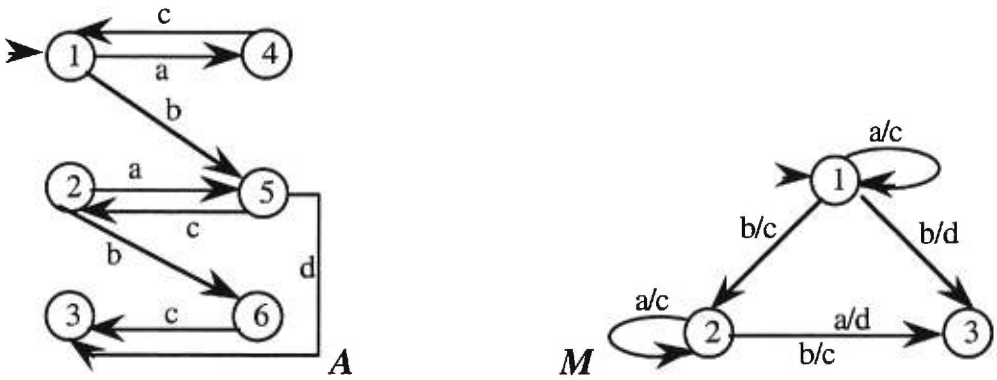


Figure 2.4 : Un automate à entrées et sorties A et la machine de Mealy M correspondante

Chaque automate à entrées et sorties A dans la classe décrite précédemment peut être transformé en une machine de Mealy ayant le même ensemble de trace que A (Figure 2.4). La machine de Mealy correspondante à l'automate à entrées et sorties $A=(S_A, I_A, O_A, T_A,$

s_{oA}) est définie par $M=(S, X, Y, h, s_o)$ où :

- $S=S_{in}$,
- $X=I_A$,
- $Y=O_A$,
- $D_A=(\cup_{s\in S_{in}}(\{s\}\times inp(s)))$,
- $\forall(s, x)\in D_A\ h(s, x)=\{(s', y)|\ \exists\ s''\in S_{out}$ tel que $(s, x, s'')\in T_A$ et $(s'', y, s')\in T_A\}$,
- $s_o=s_{oA}$.

De même, une machine de Mealy $M=(S, X, Y, h, s_o)$ peut être transformée en un automate à entrées et sorties $B=(S_B, I_B, O_B, T_B, s_{oB})$, noté $IOA(M)$, ayant le même ensemble de trace que M (Figure 2.5). L'automate à entrées et sorties B appartient à la classe décrite précédemment. La transformation est réalisée de la façon suivante :

- $S_B=S\cup D_M$,
- $I_B=X$,
- $O_B=Y$,
- $T_B=(\cup_{(s, x)\in D_M}(T(s, x))$ où $T(s, x)$ est défini comme suit :

  $\forall(s, x)\in D_M$ si $h(s, x)=\{(s_1, y_1), (s_2, y_2), ..., (s_n, y_n)\}$ alors

  $T(s, x)=\{(s, x, (s, x)), ((s, x), y_1, s_1), ((s, x), y_2, s_2), ..., ((s, x), y_n, s_n)\}$
- $s_{oB}=s_o$.

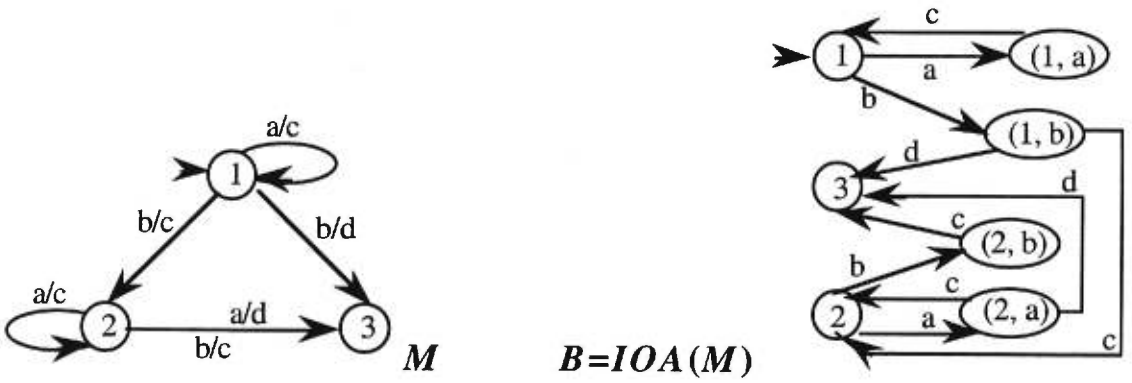


Figure 2.5 : L'automate à entrées et sorties B obtenu à partir de la machine de Mealy M

Nous pouvons remarquer que les automates à entrées et sorties A (Figure 2.4) et B (Figure 2.5) sont équivalents. Si l'on minimise B on obtient A.

Cette double correspondance permet de considérer les machines de Mealy comme une sous classe des automates à entrées et sorties.

CHAPITRE 3

LES OPÉRATEURS ET LES RELATIONS DE CONFORMITÉ

En générale, les systèmes complexes sont formés de sous systèmes plus simples qui communiquent entre eux et avec l'environnement. Chaque sous système est modélisé par un système à transitions. Il est donc impératif de disposer d'un opérateur de composition si l'on désire modéliser le système global par un système à transitions. Une fois qu'un système a été mis sous la forme d'un système à transitions, éventuellement obtenu grâce à l'opérateur de composition, on dispose d'une description formelle de ce système. On peut alors s'intéresser à certaines de ses propriétés qui s'exprimeront en termes de propriétés du système à transitions le représentant, et plus précisément en termes de propriétés de ses états, de ses transitions ou de ses chemins. Par ailleurs, un autre aspect est celui de la comparaison. Pour deux systèmes donnés, décrits dans un modèle particulier, peut-on considérer qu'un système est conforme à l'autre? Pour cela, il faut définir les critères à prendre en considération pour affirmer la conformité et définir formellement les relations de conformité.

Dans ce qui suit, nous définissons pour chacun des modèles du chapitre précédant un opérateur de composition, des relations de conformité ainsi que certaines transformations permettant de passer d'un système à transitions à un autre sous une forme jugée plus adéquate.

3.1 - Systèmes de transitions étiquetées

3.1.1 La composition des systèmes de transitions étiquetées

Les systèmes de transitions étiquetées communiquent par un mécanisme de rendez-vous négocié. Les composantes communicantes doivent exécuter une même action

simultanément. Ce type de synchronisation est celui utilisé dans CSP [Hoare 85]. La communication est bloquante dans le sens où si une composante impliquée dans la communication n'est pas dans un état où l'action commune est possible, la réalisation de cette action sera bloquée. Basé sur ce qui précède, un opérateur de composition parallèle est défini sur les systèmes de transitions étiquetées de la façon suivante :

Définition 3.1 : La composition parallèle de deux systèmes de transitions étiquetées $E_1=(S_1, L_1, T_1, s_{o1})$ et $E_2=(S_2, L_2, T_2, s_{o2})$, notée $E_1||E_2$, est définie comme la composante connexe du système de transitions étiquetées $E=(S, L, T, s_o)$ où :

- $S = S_1 \times S_2$,
- $L = L_1 \cup L_2$,
- $s_o = (s_{o1}, s_{o2})$,
- la relation de transition T de $E_1||E_2$ est obtenue au moyen des règles suivantes :
  - $a \in (L_1 \setminus L_2) \cup \{\tau\} \wedge (s_1, a, s'_1) \in T_1 \Rightarrow \forall s_2 \in S_2 , ((s_1, s_2), a, (s'_1 , s_2)) \in T$
  - $a \in (L_2 \setminus L_1) \cup \{\tau\} \wedge (s_2, a, s'_2) \in T_2 \Rightarrow \forall s_1 \in S_1 , ((s_1, s_2), a, (s_1 , s'_2)) \in T$
  - $a \in L_1 \cap L_2 \wedge (s_1, a , s'_1) \in T_1 \wedge (s_2, a , s'_2) \in T_2 \Rightarrow ((s_1, s_2), a , (s'_1 , s'_2)) \in T$

3.1.2 - Les relations de conformité

3.1.2.1 Équivalence de traces

Définition 3.2 : Soient $E_1=(S_1, L_1, T_1, s_{o1})$ et $E_2=(S_2, L_2, T_2, s_{o2})$ deux systèmes de transitions étiquetées. Deux états $s_1 \in S_1$ et $s_2 \in S_2$ sont dit équivalents par les traces, noté $s_1 \cong s_2$, lorsque $Tr_{E_1}(s_1)=Tr_{E_2}(s_2)$, sinon ils sont séparables.

Définition 3.3 : Deux systèmes de transitions étiquetées $E_1=(S_1, L_1, T_1, s_{o1})$ et $E_2=(S_2, L_2, T_2, s_{o2})$ sont dit équivalents par les traces, noté $E_1 \cong E_2$, si et seulement si $Tr_{E_1}=Tr_{E_2}$, c'est à dire lorsque leurs états initiaux sont équivalents par les traces.

L'équivalence de traces est la relation la plus simple.

3.1.2.2 Bisimulation

Lorsqu'on prend en considération les états des systèmes, la relation de bisimulation forte [Park 81] ainsi que la relation de bisimulation faible (ou équivalence observationnelle) [Milner 80] peuvent être définies.

3.1.2.2.1 Bisimulation forte

Définition 3.4 : Soient $E_1=(S_1, L_1, T_1, s_{o1})$ et $E_2=(S_2, L_2, T_2, s_{o2})$ deux systèmes de transitions étiquetées avec $L_1=L_2$. Une relation $\Re \subseteq S_1 \times S_2$ est une bisimulation forte si et seulement si

$(\forall (s_1, s_2) \in \Re)\ (\forall a \in L_1 \cup \{\tau\})$

$((s_1, a, s'_1) \in T_1 \Rightarrow \exists s'_2 |\ (s_2, a, s'_2) \in T_2 \wedge (s'_1, s'_2) \in \Re) \wedge$

$((s_2, a, s'_2) \in T_2 \Rightarrow \exists s'_1 |\ (s_1, a, s'_1) \in T_1 \wedge (s'_1, s'_2) \in \Re).$

Deux états sont dits fortement bisimilaires lorsqu'ils ont la possibilité d'executer les mêmes actions, et que chaque état atteint après l'execution d'une action à partir de l'un des deux états est fortement bisimilaire à un état atteint après l'execution de la même action à partir de l'autre état. Deux systèmes de transitions étiquetées $E_1=(S_1, L_1, T_1, s_{o1})$ et $E_2=(S_2, L_2, T_2, s_{o2})$ sont dits fortement bisimilaires, on note $E_1 \sim E_2$, si et seulement si il existe une bisimulation forte $\Re$ telle que $(s_{o1}, s_{o2}) \in \Re$.

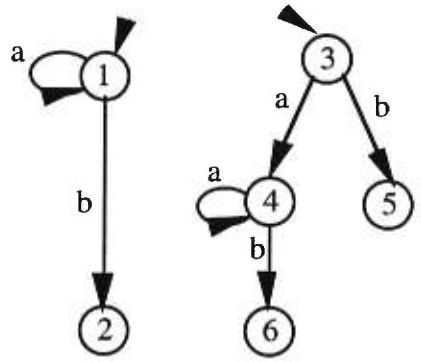


Figure 3.1 : Deux systèmes de transitions étiquetées fortement bisimilaires

3.1.2.2.2 Bisimulation faible

Si l'on considère uniquement les évolutions observables des systèmes, on obtient la relation de bissimulation faible.

Définition 3.5 : Soient $E_1=(S_1, L_1, T_1, s_{o1})$ et $E_2=(S_2, L_2, T_2, s_{o2})$ deux systèmes de transitions étiquetées avec $L_1=L_2$. Une relation $\Re \subseteq S_1 \times S$ est une bisimulation faible si et seulement si

$(\forall (s_1, s_2) \in \Re)$

$(a \in L_1) \wedge (s_1, a, s'_1) \in T_1 \Rightarrow \exists s'_2 |\ (s_2, \vec{a}, s'_2) \in T_2 \wedge (s'_1, s'_2) \in \Re \wedge$

$(s_1, \tau, s'_1) \in T_1 \Rightarrow \exists s'_2 |\ (s_2, \varepsilon, s'_2) \in T_2 \wedge (s'_1, s'_2) \in \Re \wedge$

$(a \in L_1) \wedge (s_2, a, s'_2) \in T_2 \Rightarrow \exists s'_1 |\ (s_1, \vec{a}, s'_1) \in T_1 \wedge (s'_1, s'_2) \in \Re \wedge$

$(s_2, \tau, s'_2) \in T_2 \Rightarrow \exists s'_1 |\ (s_1, \varepsilon, s'_1) \in T_1 \wedge (s'_1, s'_2) \in \Re.$

Deux systèmes de transitions étiquetées $E_1=(S_1, L_1, T_1, s_{o1})$ et $E_2=(S_2, L_2, T_2, s_{o2})$ sont dits faiblement bisimilaires (ou observationnellement équivalents), noté $E_1 \approx E_2$, si et seulement si il existe une bisimulation faible $\Re$ telle que $(s_{o1}, s_{o2}) \in \Re$.

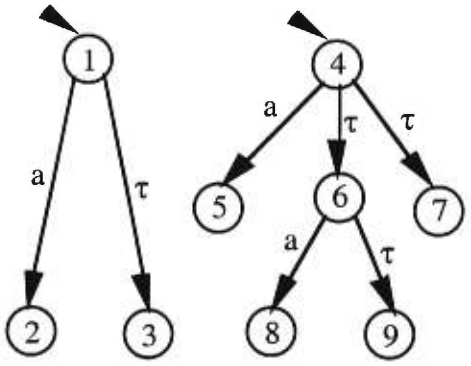


Figure 3.2 : Deux systèmes de transitions étiquetées faiblement bisimilaires

3.1.3 Transformations dans les systèmes de transitions étiquetées

Dans les systèmes de transitions étiquetées plusieurs classes peuvent être définies. On peut considérer la classe des systèmes déterministes et la classes des systèmes non-déterministes. Si l'on s'intéresse seulement aux traces du système, les classes précédantes sont équivalentes, c'est à dire, pour tout système de transitions étiquetées non-déterministe E_1 on peut trouver un système de transitions étiquetées déterministe E_2 tels que $Tr_{E_1}=Tr_{E_2}$. Cette construction est similaire à la déterminisation d'un automate non-déterministe (théorème 2.1 de [Hopcroft 87]). De plus, pour un système de transitions étiquetées déterministe, il peut être intéressant de trouver un système de transitions étiquetées déterministe ayant le même ensemble de traces et le nombre minimal d'états.

3.1.3.1 Équivalent déterministe

Définition 3.6 : L'automate de traces d'un système de transitions étiquetées $E=(S, L, T, s_o)$, noté $TM(E)$, est défini comme la composante connexe du système de transitions étiquetées (S_t, L_t, T_t, s_{to}) où :

- S_t est l'ensemble des parties de S,
- $L_t = L$,
- $s_{to} = \{s \in S \mid (s_o, \varepsilon, s)\}$,
- les transitions de l'automate sont obtenues de la façon suivante : pour tout $s \in S_t$ et $\mu \in L_t$ soit $s' = \{s_j \in S \mid (\exists s_i \in s)(s_i, \vec{\mu}, s_j)\}$, si $s' \neq \emptyset$ alors $(s, \mu, s') \in T_t$.

L'automate des traces ainsi obtenu a les mêmes traces que le système de transitions étiquetées initial, c'est à dire $Tr_{TM(E)}=Tr_E$, de plus il ne contient pas d'action interne et il est déterministe.

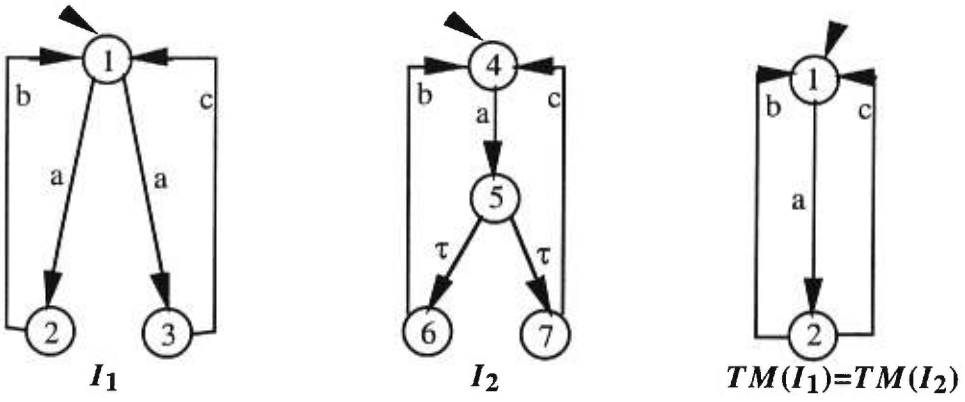


Figure 3.3 : Automates de traces

3.1.3.2 Système de transitions étiquetées minimal

Un système de transitions étiquetées déterministe est minimal lorsque tous ses états sont deux à deux séparables. Étant donné un système de transitions étiquetées déterministe, il est toujours possible de construire un système de transitions étiquetées déterministe minimal qui lui est équivalents par les traces [Wood 87].

La construction est basée sur une partition des états du système telle que deux états équivalents par les traces appartiennent à la même partie. La partition est obtenue grâce à l'utilisation des relations d'équivalence suivantes :

$s_1 \cong_k s_2$ si et seulement si s_1 et s_2 ont les mêmes traces de longueur $k \geq 1$.

Ces relations d'équivalence forment une chaîne dans le sens où $s_1 \cong_{k+1} s_2$ implique $s_1 \cong_k s_2$. Ceci implique que toute classe d'équivalence de la relation $\cong_{k+1}$ est incluse dans une classe d'équivalence de la relation $\cong_k$. Si l'on note n_k le nombre de classes d'équivalence de la relation $\cong_k$, la suite $(n_k)_{k\geq 1}$ est donc croissante. Elle est majorée par le nombre d'états du système puisque toute classe contient au moins un état. Ceci entraîne que $(n_k)_{k\geq 1}$ est convergente et donc il existe un plus petit entier non nul p tel que $n_p=n_{p+i}$ pour tout $i \geq 0$. Deux états ont les même traces de longueur k+1 signifie qu'ils ont les même traces de longueur 1 et que leurs successeurs après l'exécution d'une même action ont les même traces de longueur k. Si $n_k=n_{k+1}$, alors les successeurs ont les même traces de longueur k+1 et donc les deux états ont les même traces de longueur k+2. Ceci entraîne que $n_k=n_{k+1}$ implique $n_{k+1}=n_{k+2}$, et donc p est inférieur au nombre d'états du système. De plus, on a $s_1 \cong s_2$ si et

seulement si $s_1 \cong_k s_2$ pour tout k≥1, donc on a $s_1 \cong s_2$ si et seulement si $s_1 \cong_p s_2$. Chaque partie dans la partition recherchée est donc une classe d'équivalence de la relation $\cong_p$.

Ce qui précède permet de développer un algorithme pour la minimisation. Considérons un système de transitions étiquetées déterministe $E = (S, L, T, s_0)$. Chaque classe pour la relation $\cong_1$ contient les états dont les ensembles d'actions (traces de longueur 1) sont identiques. Par la suite, la partition associée à la relation $\cong_{k+1}$ pour 1≤k, est obtenu à partir de celle associée à la relation $\cong_k$. Deux états sont dans la même classe pour $\cong_{k+1}$, si ils sont dans la même classe pour $\cong_k$ et que leurs successeurs après l'exécution d'une même action sont aussi dans la même classe pour $\cong_k$. La partition désirée est obtenue dès que celle associée à $\cong_{k+1}$ est identique à celle associée à $\cong_k$. À partir de cette dernière, et si l'on note $[s]$ la classe d'équivalence de l'état s et $S_{\cong}$ l'ensemble des classes d'équivalence, le système de transitions étiquetées déterministe minimal associé à E est $MIN(E) = (S_M, L_M, T_M, s_{M0})$ où:

- $S_M = S_{\cong}$,
- $L_M = L$,
- $([s_1], a, [s_2]) \in T_M$ si et seulement si $(s_1, a, s_2) \in T$,
- $s_{M0} = [s_0]$.

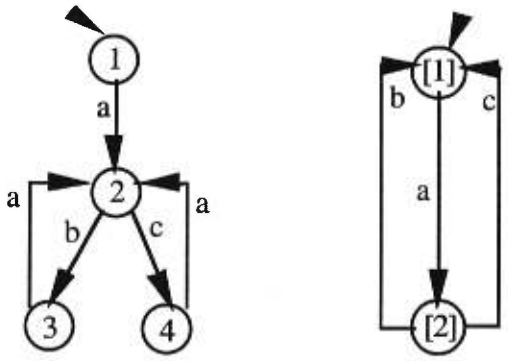


Figure 3.4 : Un système de transitions étiquetées déterministe et son équivalent minimal

3.2 Les automates à entrées et sorties

3.2.1 La composition d'automates à entrées et sorties

La composition d'automates à entrées et sorties est basée sur la synchronisation d'une sortie d'une composante avec l'entrée correspondante dans les autres composantes. Elle est définie dans le cas d'automates complètement spécifiés dans [Lynch 88]. Dans le cas d'automates partiellement spécifiés des problèmes due aux réceptions non spécifiées peuvent

apparaître lorsqu'un automate émet une sortie et que l'un des automates receveurs n'a pas de transition étiquetée par l'entrée correspondante dans son état actuel. Les composantes sont considérées autonomes, c'est à dire, une composante peut produire une sortie étiquetée par une action o sans se soucier si les composantes dont les alphabets des entrées contiennent l'action o sont dans des états où l'action o est présente. Deux approches sont alors possibles pour définir un opérateur de composition. La première consisterait à définir la composition seulement si le problème des réceptions non spécifiées n'est pas posé. Par contre, dans la deuxième approche un opérateur de composition sera défini indépendamment du problème des réceptions non spécifiées, et une vérification ultérieure sera dédié à ce problème. Nous optons pour la deuxième approche qui est celle utilisée dans [Negulescu 95][Kelekar 94].

Définition 3.7 : La composition de deux automates à entrées et sorties $B_1=(S_{B_1}, I_{B_1}, O_{B_1}, T_{B_1}, s_{o1})$ et $B_2=(S_{B_2}, I_{B_2}, O_{B_2}, T_{B_2}, s_{o2})$, telles que $O_{B_1} \cap O_{B_2}=\emptyset$, notée $B_1 \| B_2$, est définie comme la composante connexe de l'automate à entrées et sorties $B=(S_B, I_B, O_B, T_B, s_{oB})$ où:

- $S_B=S_{B_1} \times S_{B_2}$,
- $I_B=(I_{B_1} \cup I_{B_2}) \setminus (O_{B_1} \cup O_{B_2})$,
- $O_B=O_{B_1} \cup O_{B_2}$,
- $((s_1, s_2), u, (s_1', s_2')) \in T_B$ si et seulement si pour tout $i \in \{1, 2\}$, si $u \in (I_{B_i} \cup O_{B_i})$ alors $(s_i, u, s_i') \in T_{B_i}$, sinon $s_i=s_i'$,
- $s_{oB}=(s_{o1}, s_{o2})$.

La composition d'automates à entrées et sorties est commutative et associative. Elle permet qu'un nombre quelconque d'automates à entrées et sorties acceptent la même entrée simultanément.

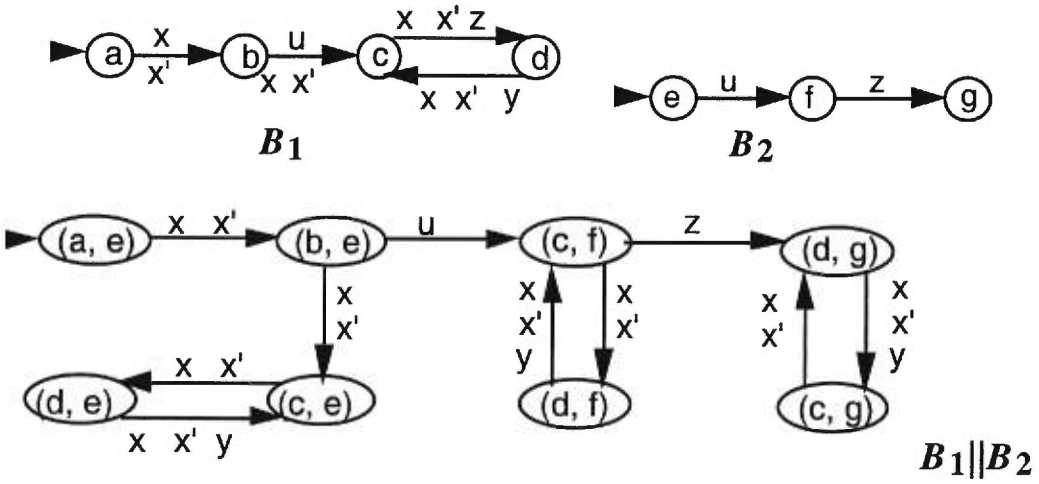


Figure 3.5 : La composition d'automates à entrées et sorties

Nous illustrons la composition d'automates à entrées et sorties par l'exemple de la Figure 3.5. Nous considérons les automates à entrées et sorties B_1 et B_2 avec $I_{B_1}=\{x, x', z\}$, $O_{B_1}=\{u, y\}$, $I_{B_2}=\{u\}$ et $O_{B_2}=\{z\}$. Pour la composition $B=B_1||B_2$ nous obtenons $I_B=\{x, x'\}$ et $O_B=\{u, z, y\}$.

De manière analogue au travail de [Negulescu 95], nous définissons une propriété de sécurité afin de formaliser la non existence de réceptions non spécifiées dans la composition B d'une collection d'automates à entrées et sorties $(B_i=(S_{B_i}, I_{B_i}, O_{B_i}, T_{B_i}, s_{oi}))_{1\leq i\leq n}$. On note ε le mot vide.

Définition 3.8 : Étant donnée une collection d'automates à entrées et sorties $(B_i=(S_{B_i}, I_{B_i}, O_{B_i}, T_{B_i}, s_{oi}))_{1\leq i\leq n}$, la composition $B=B_1||B_2||...||B_n$ est sécuritaire, noté $\mathbb{S}(B)$, si et seulement si tout mot t dans $(I_B\cup O_B)^*$ tel que $Pr_{B_i}(t)\in Tr_{B_i}.(I_{B_i}\cup\{\varepsilon\})$ pour tout i, est une trace de A (c'est à dire $t\in Tr_B$).

Cette définition regroupe deux propriétés importantes. La première propriété garantie que dans tout état de la composition B, si une composante est dans un état où elle peut produire une sortie étiquetée par une action o alors toutes les composantes dont les alphabets des entrées contiennent l'action o sont dans des états où l'action o est possible. La deuxième propriété assure que dans tout état de la composition B, toutes les actions appartenant à l'alphabet des entrées de B sont possibles. On remarque à ce niveau que la propriété de sécurité entraîne que la composition $B=B_1||B_2||...||B_n$ est complètement spécifiée.

Dans l'exemple de la Figure 3.5, la composition $B=B_1||B_2$ n'est pas sécuritaire car pour le mot $t=xux'z$ dans $(I_B\cup O_B)^*$, nous avons $Pr_{B_1}(t)\in Tr_{B_1}.(I_{B_1})$ et $Pr_{B_2}(t)\in Tr_{B_2}$, mais $t\notin Tr_B$.

Proposition 3.1 : Étant donnée une collection d'automates à entrées et sorties $(B_i=(S_{B_i}, I_{B_i}, O_{B_i}, T_{B_i}, s_{oi}))_{1\leq i\leq n}$, les propositions suivantes sont équivalentes :

i - la composition $B=B_1||B_2||...||B_n$ est sécuritaire,

ii - l'état *Fail* n'est pas accessible dans $C=Ief(B_1)||Ief(B_2)||...||Ief(B_n)$.

Preuve de Proposition 3.1 :

Première partie : (i) $\Rightarrow$ (ii)

Supposons que l'état *Fail* est accessible dans $C=Ief(B_1)||Ief(B_2)||...||Ief(B_n)$,

Donc il existe un mot $t=t'.x$ dans $(I_B\cup O_B)^*.(I_{B_1}\cup I_{B_2}\cup ...\cup I_{B_n})$ tel que $(s_{oC})_t=Fail$,

Ceci entraîne que $Pr_{B_i}(t)\in Tr_{B_i}.(I_{B_i}\cup\{\varepsilon\})$ pour tout i, mais $t\notin Tr_B$,

Ceci est en contradiction avec le faite que B est sécuritaire,

Nous concluons que l'état *Fail* n'est pas accessible dans C.

Deuxième partie : (ii) $\Rightarrow$ (i)

Soit t un mot dans $(I_B \cup O_B)^*$ tel que $Pr_{B_i}(t) \in Tr_{B_i}.(I_{B_i} \cup \{\varepsilon\})$ pour tout i,

D'après la définition de la composition $t \in Tr_C$,

Puisque l'état *Fail* n'est pas accessible dans C, ceci entraîne que $t \in Tr_B$,

Nous concluons que la composition $B=B_1||B_2||...||B_n$ est sécuritaire.

La Proposition 3.1 permet de façon pratique la vérification de la propriété de sécurité pour une composition d'automates à entrées et sorties. La Figure 3.6 illustre l'application de la Proposition 3.1 à la composition des automates B_1 et B_2 de la Figure 3.5.

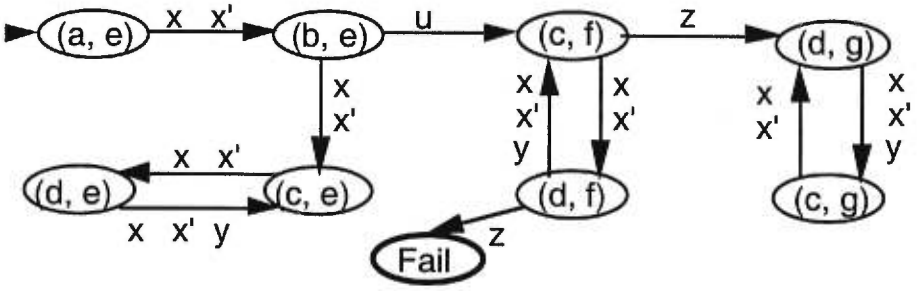


Figure 3.6 : La composition $C=Ief(B_1)||Ief(B_2)$

3.2.2 Les relations de conformité

Une relation de conformité $\Re_1$ est plus fine qu'une relation de conformité $\Re_2$ lorsque pour toute paire d'automates à entrées et sortie A et B, $A\Re_1 B$ implique $A\Re_2 B$.

Nous donnons d'abord les définitions de certaines relations de conformité bien connues. Par la suite, nous définissons la réalisation sécuritaire qui sera la relation de conformité la moins fine. Finalement, nous définissons la relation de conformité sous-type qui sera une relation générique dans le sens où toutes les relations décrites pourront être considérées comme des cas particuliers.

3.2.2.1 Équivalence de traces

Deux automates à entrées et sorties A et B satisfont la relation d'équivalence de traces, noté $B \cong A$, si et seulement si $Tr_A = Tr_B$ [Glabbeek 90].

Dans le cas des automates à entrées et sorties déterministes, les relations de

conformité bissimulation forte et bissimulation faible se confondent avec l'équivalence de traces.

3.2.2.2 Quasi-équivalence

L'automate à entrées et sorties déterministe B est quasi-équivalent à l'automate à entrées et sorties déterministe A, noté $B \leq_{qe} A$, si et seulement si

$$\forall \sigma \in Tr_A \ (\sigma \in Tr_B \wedge out((s_{oA})_\sigma) = out((s_{oB})_\sigma)).$$

La relation quasi-équivalence exige que $Tr_A \subseteq Tr_B$ et après chaque trace dans Tr_A, A et B doivent produire le même ensemble de sorties [Phalippou 94][Luo 93]. Elle est moins fine que la relation d'équivalence de traces.

3.2.2.3 Réduction

L'automate à entrées et sorties déterministe B est une réduction de l'automate à entrées et sorties déterministe A, noté $B \leq_{red} A$, si et seulement si pour chaque trace σ dans Tr_A si σ est dans Tr_B alors :

$$inp((s_{oA})_\sigma) \subseteq inp((s_{oB})_\sigma) \wedge out((s_{oB})_\sigma) \subseteq out((s_{oA})_\sigma) \wedge (out((s_{oA})_\sigma) \neq \emptyset \Rightarrow out((s_{oB})_\sigma) \neq \emptyset).$$

La relation quasi-équivalence est plus fine que la relation réduction.

3.2.2.4 La réalisation sécuritaire

Afin qu'une machine modélisée par un automate à entrées et sorties fonctionne correctement dans un environnement donné, elle doit au moins accepter toutes les entrées soumises par l'environnement et ne produire que des sorties acceptées par l'environnement. Ceci signifie qu'il n'y a pas de réceptions non spécifiées dans la composition des automates à entrées et sorties modélisant la machine et l'environnement. Cette propriété mène à la définition de la relation de conformité réalisation sécuritaire.

Définition 3.9 : Considérons un automate à entrées et sorties A et un automate à entrées et sorties $B=B_1||B_2||...||B_n$ tel que $I_A=I_B$. L'automate B est une réalisation sécuritaire de A, noté $B \leq_{\mathbb{S}} A$, si et seulement si pour tout automate à entrées et sorties E, tel que $I_E=O_A$ and $O_E=I_A$, $\mathbb{S}(E||A)$ implique $\mathbb{S}(E||B_1||B_2||...||B_n)$.

La Définition 3.9 capture le fait que pour tout environnement E, ayant les même ensembles d'entrées et de sorties que A, si la composition de A et E est sécuritaire alors la composition de B_1, B_2, ..., B_n et E doit aussi être sécuritaire, c'est à dire dans tout état

accessible de la composition $E||B_1||B_2||...||B_n$, il n'existe pas de réception non spécifiée.

Définition 3.10 : La réflexion d'un automate à entrées et sorties déterministe $A=(S_A, I_A, O_A, T_A, s_{oA})$ est l'automate à entrées et sorties $\tilde{A}=(S_A, I_{\tilde{A}}, O_{\tilde{A}}, T_A, s_{oA})$ où $I_{\tilde{A}}=O_A$ et $O_{\tilde{A}}=I_A$.

Si l'on ajoute à un état de A, une transition étiquetée par une action dans O_A alors la composition de l'automate à entrées et sorties obtenu avec $\tilde{A}$ sera non sécuritaire. De même, si l'on ajoute à un état de $\tilde{A}$, une transition étiquetée par une action dans I_A alors la composition de l'automate à entrées et sorties obtenue avec A sera non sécuritaire. Intuitivement, $\tilde{A}$ représente l'environnement le plus libéral par rapport aux sorties et le moins défini par rapport aux entrées dans lequel A est sécuritaire.

Propostition 3.2 : Pour un automate à entrées et sorties déterministe A et un automate à entrées et sorties $B=B_1||B_2||...||B_n$, tel que $I_A=I_B$, les propositions suivante sont équivalentes:

i - $\mathbb{S}(\tilde{A}||B_1||B_2||...||B_n)$,

ii - pour tout automate à entrées et sorties E, tel que $I_E=O_A$ et $O_E=I_A$,

$$\mathbb{S}(E||A) \Rightarrow \mathbb{S}(E||B_1||B_2||...||B_n).$$

Preuve de Propostition 3.2 :

Première partie : (i) $\Rightarrow$ (ii)

Soit E un automate à entrées et sorties, tel que $I_E=O_A$ et $O_E=I_A$, vérifiant $\mathbb{S}(E||A)$.

Nous devons montrer $\mathbb{S}(E||B_1||B_2||...||B_n)$.

Pour une trace σ, on note $|\sigma|$ la longueur de σ (c'est à dire le nombre d'actions dans σ) et $\sigma_{[k]}$ le préfixe de σ de longueur k.

Supposons qu'il existe $\sigma \in (I_{(E||B_1||B_2||...||B_n)} \cup O_{(E||B_1||B_2||...||B_n)})^*$ avec $|\sigma|=m$ telle que :

$$Pr_E(\sigma) \in Tr_E.(I_E \cup \{\varepsilon\}) \wedge Pr_{B_i}(\sigma) \in Tr_{B_i}.(I_{B_i} \cup \{\varepsilon\}) \text{ pour tout } 1 \leq i \leq n$$

Nous montrons d'abord par induction que $Pr_E(\sigma) \in Tr_A$.

Si $Pr_E(\sigma_{[1]})=\varepsilon$ alors $Pr_E(\sigma_{[1]}) \in Tr_A$

Si $Pr_E(\sigma_{[1]}) \in I_E$, puisque $\mathbb{S}(\tilde{A}||B_1||B_2||...||B_n)$ alors

$$Pr_E(\sigma_{[1]}) \in Tr_{\tilde{A}}.I_{\tilde{A}} \wedge Pr_{B_i}(\sigma_{[1]}) \in Tr_{B_i}.(I_{B_i} \cup \{\varepsilon\}) \text{ pour } 1 \leq i \leq n \Rightarrow Pr_E(\sigma_{[1]}) \in Tr_A$$

Si $Pr_E(\sigma_{[1]}) \in O_E$, puisque $\mathbb{S}(E||A)$ alors

$$Pr_E(\sigma_{[1]}) \in Tr_E \wedge Pr_A(\sigma_{[1]}) \in Tr_A.I_A \Rightarrow Pr_E(\sigma_{[1]}) \in Tr_A$$

Supposons que $Pr_E(\sigma_{[k]}) \in Tr_A$ pour $1 \leq k < m$, et posons $Pr_E(\sigma_{[k+1]})=Pr_E(\sigma_{[k]}).t$

Si $t=\varepsilon$ alors $Pr_E(\sigma_{[k+1]}) \in Tr_A$

Si $t \in I_E$, puisque $\mathbb{S}(\tilde{A}||B_1||B_2||...||B_n)$ alors

$$Pr_E(\sigma_{[k+1]}) \in Tr_{\tilde{A}}.I_{\tilde{A}} \wedge Pr_{B_i}(\sigma_{[k+1]}) \in Tr_{B_i}.(I_{B_i} \cup \{\varepsilon\}) \text{ pour } 1 \leq i \leq n \Rightarrow Pr_E(\sigma_{[k+1]}) \in Tr_A$$

Si $t \in O_E$, puisque $\mathbb{S}(E\|A)$ alors

$$Pr_E(\sigma_{[k+1]}) \in Tr_E \wedge Pr_A(\sigma_{[k+1]}) \in Tr_A.I_A \Rightarrow Pr_E(\sigma_{[k+1]}) \in Tr_A$$

D'après le principe d'induction $Pr_E(\sigma) \in Tr_A$.

Puisque $\mathbb{S}(\tilde{A}\|B_1\|B_2\|...\|B_n)$ alors

$Pr_E(\sigma) \in Tr_A \wedge Pr_{B_i}(\sigma) \in Tr_{B_i}.(I_{B_i} \cup \{\varepsilon\})$ pour $1 \leq i \leq n \Rightarrow Pr_{B_i}(\sigma) \in Tr_{B_i}$ pour $1 \leq i \leq n$

Puisque $\mathbb{S}(E\|A)$ alors $Pr_E(\sigma) \in Tr_E.(I_E \cup \{\varepsilon\}) \wedge Pr_E(\sigma) \in Tr_A \Rightarrow Pr_E(\sigma) \in Tr_E$

Donc $\sigma \in Tr_{(E\|B_1\|B_2\|...\|B_n)}$.

Nous concluons que $\mathbb{S}(E\|B_1\|B_2\|...\|B_n)$.

Deuxième partie : (ii) $\Rightarrow$ (i)

Cette partie est évidente car $\tilde{A}$ est un automate à entrées et sorties avec $I_{\tilde{A}}=O_A$ et $O_{\tilde{A}}=I_A$, et l'on a $\mathbb{S}(E\|A)$. □

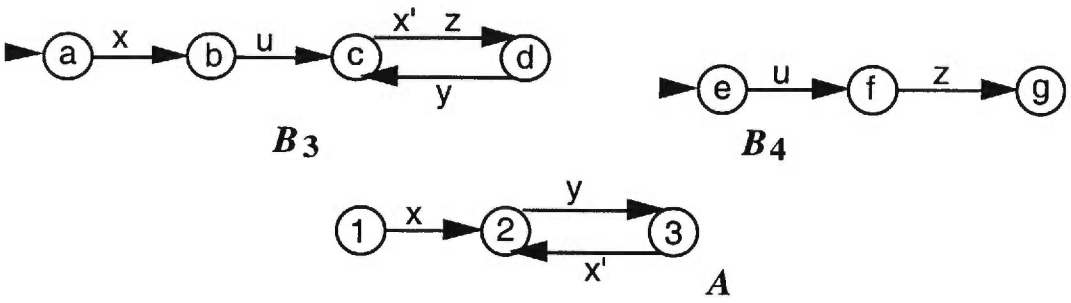


Figure 3.7 : Les automates à entrées et sorties B_3, B_4 et A

L'automate à entrées et sorties $B=B_3\|B_4$ obtenu à partir de la composition des automates B_3 et B_4 de la Figure 3.7 est une réalisation sécuritaire de l'automate à entrées et sorties A présenté dans la Figure 3.7, car la trace visible xx' qui peut mener à l'apparition d'une réception non spécifiée n'est pas permise dans A.

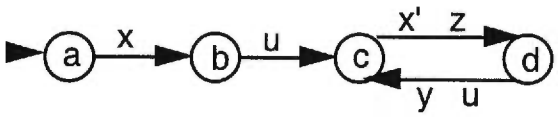


Figure 3.8 : L'automate à entrées et sorties B_5

Nous illustrons la nécessite de considérer toutes les composantes lors de la vérification de la réalisation sécuritaire d'un automate à entrées et sorties A par une composition $B=B_1\|B_2\|...\|B_n$ grâce à l'exemple suivant. On considère l'automate à entrées et sorties B_5 de la Figure 3.8 avec $I_{B_5}=I_{B_3}$ et $O_{B_5}=O_{B_3}$, on remarque que $B_3\|B_4=B_5\|B_4$ mais $B_5\|B_4$ n'est pas une réalisation sécuritaire de A (Figure 3.7) car pour le mot $t'=xuzu$ dans

$(I_B \cup O_B)^*$, nous avons $Pr_{B5}(t') \in Tr_{B5}$, $Pr_{B2}(t') \in Tr_{B4}.(I_{B4})$ et $Pr_A(t') \in Tr_A$, mais t' n'est pas une trace de la composition $\tilde{A} \,\|B_5\|B_4$. Ceci résulte du faite que lors de la définition de la composition le problème des réceptions non spécifiées n'a pas été pris en compte.

Lemme 3.1: Chacune des relations de conformité équivalence de trace, quasi-équivalence et réduction implique la réalisation sécuritaire, c'est à dire, si l'une de ces relations est satisfaite entre deux automates à entrées et sorties déterministes B et A alors B est une réalisation sécuritaire de A.

La preuve du lemme 3.1 s'obtient directement à partir des définitions. La réalisation sécuritaire est une relation de conformité moins fine que les relations de conformité équivalence de trace, quasi-équivalence et réduction.

3.2.2.5 La relation sous-type

La réalisation sécuritaire est un critère de conformité faible. Elle permet à un automate à entrées et sorties B qui est une réalisation sécuritaire d'un automate à entrées et sorties A de ne produire aucune sortie. La notion de tache complète impose qu'après une trace donnée le système doit progresser afin de produire certaines sorties. La notion de choix permet à une implantation conforme d'avoir au moins un comportement parmi un ensemble de comportements décrits par la spécification. Pour prendre en compte les deux notions précédentes, nous définissons le modèle des automates à entrées et sorties avec traces complètes optionnelles.

Définition 3.11 : Un automate à entrées et sorties avec traces complètes optionnelles A (*IOAWOCT* : Input-Output Automaton With Optional Complete Traces), est défini par la donnée d'un triplet (IOA_A, MT_A, OCT_A) où IOA_A est un automate à entrées et sorties, $MT_A=\{(s, MT_A(s))|\ s \in S_A\}$ avec $MT_A(s) \subseteq 2^{Tr_{IOA_A}(s)}$ et $OCT_A=\{(s, OCT_A(s))|\ s \in S_A\}$ avec $OCT_A(s) \subseteq Tr_{IOA_A}(s)$.

Un automate à entrées et sorties avec traces complètes optionnelles A peut être considéré comme une spécification d'un système. Les ensembles $MT_A(s)$ et $OCT_A(s)$ imposent des contraintes sur les traces des implantations valides de A. Les implantations sont décrites par des automates à entrées et sorties. Un élément Y de $MT_A(s)$ impose qu'au moins une trace dans Y, est égale à la projection sur l'alphabet de IOA_A d'une trace possible de l'implantation dans l'état correspondant à s. De plus, chaque fois qu'une exécution débutant dans l'état correspondant à s dans l'implantation a une trace qui est un préfixe d'un élément

de $OCT_A(s)$, cette exécution doit progresser afin de compléter une trace dont la projection sur l'alphabet de IOA_A est dans $OCT_A(s)$. On note que si $OCT_A(s)$ contient seulement des éléments de longueur égale à un, il n'impose aucune contrainte sur les implantations ayant le même alphabet que IOA_A. Dans le cas où l'alphabet d'une implantation B est différent de l'alphabet de IOA_A, les éléments de longueur égale à un dans $OCT_A(s)$ interdisent les traces silencieuses dans l'ensemble des traces de l'état s' correspondant à s dans B.Une trace dans $Tr_B(s')$ est dite silencieuse si elle est formée seulement par des actions n'appartenant pas à l'alphabet de A et que elle n'est pas préfixe d'au moins une trace dans $Tr_B(s')$ dont la projection sur l'alphabet de IOA_A est un élément de O_{IOA_A}.

Un automate à entrées et sorties A peut être considéré comme un automate à entrées et sorties avec traces complètes optionnelles, que nous notons A^{woct}. Il est construit de la façon suivante : $IOA_A\text{woct}=A$ et pour tout état s de A $MT_A\text{woct}(s)=\{\{y\}|\ y\in out(s)\}$ et $OCT_A\text{woct}(s)=out(s)$. Cette application de l'ensemble des automates à entrées et sorties vers l'ensemble des automates à entrées et sorties avec traces complètes optionnelles est injective. Elle permet d'inclure le premier ensemble dans le second ensemble.

Dans la définition suivante, nous formalisons la propriété de progrès pour un automate à entrées et sorties. Pour un ensemble de traces X, on note $Pref(X)$ l'ensemble de tous les préfixes de longueur non nulle des éléments de X.

Définition 3.12 : Étant donné un automate à entrées et sorties avec traces complètes optionnelles déterministe A, l'automate à entrées et sorties déterministe B tel que $I_B=I_{IOA_A}$ réalise la propriété de progrès par rapport à $IOAWOCT\,A$, noté $B \leq_{\mathbb{P}} A$, si et seulement si

Si $MT_A(s_{0IOA_A})\neq\emptyset$ alors

i - Pour tout X dans $MT_A(s_{0IOA_A})$, $Pr_{IOA_A}(Tr_B(s_{0B}))\cap X\neq\emptyset$,

ii - Pour toute trace $\sigma_1\in Tr_B(s_{0B})$, si $Pr_{IOA_A}(\sigma_1)\in Pref(OCT_A(s_{0IOA_A}))$ alors il existe une trace $\sigma_2\in Tr_B((s_{0B})_{\sigma_1})$ telle que $Pr_{IOA_A}(\sigma_1\sigma_2)\in OCT_A(s_{0IOA_A})$.

Et, pour toute trace $\sigma t\in Tr_B$ avec $t\in(I_{IOA_A}\cup O_{IOA_A})$, si $\sigma'=Pr_{IOA_A}(\sigma t)\in Tr_{IOA_A}$ et $MT_A((s_{0IOA_A})_{\sigma'})\neq\emptyset$ alors

iii - Pour tout X dans $MT_A((s_{0IOA_A})_{\sigma'})$, $Pr_{IOA_A}(Tr_B((s_{0B})_{\sigma t}))\cap X\neq\emptyset$,

iv - Pour toute trace $\sigma_1\in Tr_B((s_{0B})_{\sigma t})$, si $Pr_{IOA_A}(\sigma_1)\in Pref(OCT_A((s_{0IOA_A})_{\sigma'}))$ alors il existe une trace $\sigma_2\in Tr_B((s_{0B})_{\sigma t\sigma_1})$ telle que $Pr_{IOA_A}(\sigma_1\sigma_2)\in OCT_A((s_{0IOA_A})_{\sigma'})$.

Les conditions (i) et (iii) imposent qu'au moins une trace dans chaque élément de $MT_A((s_{0IOA_A})_{\sigma'})$ est présente dans la projection de $Tr_B((s_{0B})_{\sigma t})$ sur l'alphabet de IOA_A. Les conditions (ii) et (iv) imposent que pour chaque trace dans $Tr_B((s_{0B})_{\sigma t})$ dont la projection sur

l'alphabet de IOA_A est un préfixe d'un élément de $OCT_A((s_{0IOA_A})_{\sigma'})$, il existe une trace dans $Tr_B((s_{0B})_{\sigma t})$ dont la projection sur l'alphabet de IOA_A est un élément de $OCT_A((s_{0IOA_A})_{\sigma'})$.

Dans les langages orienté-objet, la notion de sous-type, qui est une relation de conformité entre types, est définie. Un type P est un sous-type d'un autre type Q si P offre au moins les opérations offertes par Q (P peut aussi offrir des opérations additionnelles). De plus, les types des résultats des opérations dans P' doivent être des sous-types des types des résultats des opérations correspondantes dans Q. Finalement, les types des arguments des opérations dans Q doivent être des sous types de ceux des opérations correspondantes dans P' [Black 87]. La notion de sous-type permet l'utilisation d'une instance d'un sous-type d'un type T chaque fois qu'une instance du type T est requise pour la réalisation d'une tache donnée.

Alors que la relation sous-type dans les langages orienté-objet s'intéresse principalement aux opérations disponibles ainsi qu'aux types de leurs paramètres [America 85][Halbert 86], nous sommes intéressé par le comportement dynamique des objets modélisés par des automates à entrées et sorties, en considérant les séquences permises d'entrées et de sorties. Nous définirons une relation sous-type, notée $\not\delta$, dans le même but que celle des langages orienté-objet, c'est-à-dire, la possibilité de remplacer n'importe quel sous système par une instance de ses sous-types tout en maintenant la conformité du système par rapport à sa spécification.

Considérons un automate à entrées et sorties avec traces complètes optionnelles A et un environnement E tels que $\mathbb{S}(E||IOA_A)$. Si l'on désire remplacer IOA_A dans l'environnement E par un automate à entrées et sorties B qui est une instance d'un sous-type de A, B doit être une réalisation sécuritaire de IOA_A, c'est à dire $B \leq_{\mathbb{S}} IOA_A$, de plus, puisque pour chaque état s de IOA_A, il y a des conditions sur l'ensemble $Tr_{IOA_A}(s)$, B doit satisfaire ces conditions. Un type est décrit par un automate à entrées et sorties avec traces complètes optionnelles, alors qu'une instance est décrites par un automate à entrées et sorties. Dans ce but, on défini une nouvelle relation de conformité nommée implantation conforme. Cette relation requiert la satisfaction de la réalisation sécuritaire ainsi que la propriété de progrès.

Définition 3.13 : Étant donné un automate à entrées et sorties avec traces complètes optionnelles déterministe A, l'automate à entrées et sorties déterministe B tel que $I_B=I_{IOA_A}$, est une implantation conforme de A, noté $B \leq_{conf} A$, si et seulement si $B \leq_{\mathbb{P}} A$ et $B \leq_{\mathbb{S}} IOA_A$.

La relation sous-type entre deux automates à entrées et sorties avec traces complètes optionnelles déterministes capture le faite que toutes les implantations conformes du sous-type sont aussi des implantations conformes du type.

Définition 3.14 : Un automate à entrées et sorties avec traces complètes optionnelles déterministe B est un sous-type d'un automate à entrées et sorties avec traces complètes optionnelles déterministe A, noté $B \not\& A$, si pour tout automate à entrées et sorties déterministe C : $C \leq_{conf} B$ implique $C \leq_{conf} A$.

Lemme 3.2 : Étant donnés un automate à entrées et sorties avec traces complètes optionnelles déterministe A et un automate à entrées et sorties déterministe B tels que $I_B = I_{IOA_A}$, les propositions suivantes sont équivalentes :

i - $B \leq_{conf} A$ ii - $B^{woct} \not\& A$.

La preuves du lemme 3.2 s'obtient directement à partir des définitions.

Les relations de conformité équivalence de trace, quasi-équivalence et réduction peuvent être considérées comme des cas particulier de la relation implantation conforme.

Lemme 3.3 : Étant donnés deux automates à entrées et sorties déterministes A et B tels que $I_A = I_B$ et $O_A = O_B$. Les propositions suivantes sont équivalentes :

i - $B \leq_{qe} A$,

ii - $B \leq_{conf} A^{woct}$.

Preuve de Lemme 3.3 :

Première partie : (i) $\Rightarrow$ (ii)

Montrons d'abord que $B \leq_{\S} A$

D'après Proposition 3.1 et Proposition 3.2 ceci revient à montrer que l'état *Fail* n'est pas accessible dans $C = Ief(B) \| Ief(\tilde{A})$

Supposons que : $\exists \sigma \in (I_B \cup O_B)^*$ / $(s_{oC})_\sigma = Fail$ et posons $\sigma = \sigma_1.t$

Si $t \in I_A$ alors $\sigma \in Tr_A$ et comme $B \leq_{qe} A$ donc $\sigma \in Tr_B$

Si $t \in O_A$ alors $\sigma \in Tr_B$ or $\sigma_1 \in Tr_A$ et $t \in out((s_{oB})_{\sigma_1}) = out((s_{oA})_{\sigma_1})$ donc $\sigma \in Tr_A$

Ceci contredit le faite que σ mène à l'état *Fail* dans C

Nous concluons que $B \leq_{\S} A$

Montrons Maintenant que $B \leq_{\mathbb{P}} A^{woct}$

Comme A et B ont les mêmes alphabets d'entrées et de sorties et vue la définition de A^{woct}, il suffit de vérifier (i) et (iii) dans Définition 3.12

Vérification de (i) :

Si $MT_A(s_{0A})\neq\emptyset$ alors tout X dans $MT_A(s_{0A})$ est inclus dans $out(s_{0A})=out(s_{0B})$ et donc $Tr_B\cap X\neq\emptyset$

Vérification de (iii) :

Soit $\sigma\in Tr_B$ si $\sigma\in Tr_A$ et $MT_A((s_{0A})_\sigma)\neq\emptyset$ alors tout X dans $MT_A((s_{0A})_\sigma)$ est inclus dans $out((s_{0A})_\sigma)=out((s_{0B})_\sigma)$ et donc $Tr_B((s_{0B})_\sigma)\cap X\neq\emptyset$

Nous concluons que $B\leq_{\mathbb{P}}A^{woct}$

Deuxième partie : (ii) $\Rightarrow$ (i)

Pour une trace σ, on note $|\sigma|$ la longueur de σ (c'est à dire le nombre d'actions dans σ)

Nous montrons par induction sur la longueur des traces que :

$$\forall\sigma\in Tr_A\ (\sigma\in Tr_B\wedge out((s_{0A})_\sigma)=out((s_{0B})_\sigma))$$

Base d'induction : $|\sigma|=0$

Puisque $B\leq_{\mathbb{P}}A^{woct}$ alors d'après Définition 3.12 (i) on a $out(s_{0A})\subseteq out(s_{0B})$,

Puisque $B\leq_{\mathbb{S}}A$ alors d'après Proposition 3.2 on a $out(s_{0B})\subseteq out(s_{0A})$

Hypothèse d'induction : $|\sigma|=n\geq 0\ \forall\sigma\in Tr_A\ (\sigma\in Tr_B\wedge out((s_{0A})_\sigma)=out((s_{0B})_\sigma))$

Soit $|\sigma|=n+1$ avec $\sigma\in Tr_A$, on pose $\sigma=\sigma_1.t$

D'après l'hypothèse d'induction $\sigma_1\in Tr_B\wedge out((s_{0A})_{\sigma_1})=out((s_{0B})_{\sigma_1})$

Si $t\in O_A$ alors $t\in out((s_{0A})_{\sigma_1})=out((s_{0B})_{\sigma_1})$ donc $\sigma\in Tr_B$

Si $t\in I_A$ alors $B\leq_{\mathbb{S}}A$ entraîne d'après Proposition 3.2 que $\sigma\in Tr_B$

De plus,

Puisque $B\leq_{\mathbb{P}}A^{woct}$ alors d'après Définition 3.12 (iii) on a

$$out((s_{0A})_\sigma)\subseteq out((s_{0B})_\sigma),$$

Puisque $B\leq_{\mathbb{S}}A$ alors d'après Proposition 3.2 on a $out((s_{0B})_\sigma)\subseteq out((s_{0A})_\sigma)$

D'après le principe d'induction, nous avons :

$$\forall\sigma\in Tr_A\ (\sigma\in Tr_B\wedge out((s_{0A})_\sigma)=out((s_{0B})_\sigma))$$

Nous concluons que : $B\leq_{conf}A^{woct}\Rightarrow B\leq_{qe}A$. □

Lemme 3.4 : Étant donnés deux automates à entrées et sorties déterministes A et B tels que $I_A=I_B$ et $O_A=O_B$. Les propositions suivantes sont équivalentes :

i - $B\cong A$,

ii - $B\leq_{conf}A^{woct}$ et $A\leq_{conf}B^{woct}$.

Preuve de Lemme 3.4 :

$B\cong A\quad\Leftrightarrow B\leq_{qe}A\wedge A\leq_{qe}B$ d'après les définitions

$\Leftrightarrow B\leq_{conf}A^{woct}\wedge A\leq_{conf}B^{woct}$ d'après Lemme 3.3 □

Lemme 3.5: Étant donnés deux automates à entrées et sorties déterministes A et B tels que $I_A=I_B$ et $O_A=O_B$. On note A^{red} l'automate à entrées et sorties avec traces complètes optionnelles tel que $IOA_{A^{red}}=A$ et pour tout état s de A $MT_{A^{red}}(s)=\{out(s)\}$ si $out(s)\neq\emptyset$, sinon $MT_{A^{red}}(s)=\emptyset$ et $OCT_{A^{red}}(s)=out(s)$. Les propositions suivantes sont équivalentes :

i - $B \leq_{red} A$,

ii - $B \leq_{conf} A^{red}$.

Preuve de Lemme 3.5 :

Première partie : (i) $\Rightarrow$ (ii)

Montrons d'abord que $B \leq_{\mathbb{S}} A$

D'après Proposition 3.1 et Proposition 3.2 ceci revient à montrer que l'état *Fail* n'est pas accessible dans $C=Ief(B)\|Ief(\tilde{A})$

Supposons que : $\exists\sigma\in(I_B\cup O_B)^*$ / $(s_{oC})_\sigma=Fail$ et posons $\sigma=\sigma_1.t$

Si $t\in I_A$ alors $\sigma\in Tr_A$, or $\sigma_1\in Tr_B$ et $t\in inp((s_{oA})_{\sigma_1})\subseteq inp((s_{oB})_{\sigma_1})$ donc $\sigma\in Tr_B$

Si $t\in O_A$ alors $\sigma\in Tr_B$, or $\sigma_1\in Tr_A$ et $t\in out((s_{oB})_{\sigma_1})\subseteq out((s_{oA})_{\sigma_1})$ donc $\sigma\in Tr_A$

Ceci contredit le faite que σ mène à l'état *Fail* dans C

Nous concluons que $B \leq_{\mathbb{S}} A$

Montrons Maintenant que $B \leq_{\mathbb{P}} A^{red}$

Comme A et B ont les mêmes alphabets d'entrées et de sorties et vue la définition de A^{red}, il suffit de vérifier (i) et (iii) dans Définition 3.12

Vérification de (i) :

Si $MT_A(s_{oA})\neq\emptyset$ alors $MT_A(s_{oA})=\{out(s_{oA})\}$

Comme $B \leq_{red} A$ on a $out(s_{oB})\subseteq out(s_{oA}) \wedge$ et $(out(s_{oA}))\neq\emptyset \Rightarrow out(s_{oB}))\neq\emptyset)$

Donc $Tr_B\cap out(s_{oA})\neq\emptyset$

Vérification de (iii) :

Soit $\sigma\in Tr_B$ si $\sigma\in Tr_A$ et $MT_A((s_{oA})_\sigma)\neq\emptyset$ alors $MT_A((s_{oA})_\sigma)=\{out((s_{oA})_\sigma)\}$

Comme $B \leq_{red} A$ on a $out((s_{oB})_\sigma)\subseteq out((s_{oA})_\sigma)$ et $(out((s_{oA})_\sigma)\neq\emptyset \Rightarrow out((s_{oB})_\sigma)\neq\emptyset)$

Donc $Tr_B((s_{oB})_\sigma)\cap out((s_{oA})_\sigma)\neq\emptyset$

Nous concluons que $B \leq_{\mathbb{P}} A^{red}$

Deuxième partie : (ii) $\Rightarrow$ (i)

Pour une trace σ, on note $|\sigma|$ la longueur de σ (c'est à dire le nombre d'actions dans σ)

Montrons que : $\forall\sigma\in Tr_A$ $(\sigma\in Tr_B \Rightarrow inp((s_{oA})_\sigma)\subseteq inp((s_{oB})_\sigma)$

$\wedge\ out((s_{oB})_\sigma)\subseteq out((s_{oA})_\sigma) \wedge (out((s_{oA})_\sigma)\neq\emptyset \Rightarrow out((s_{oB})_\sigma)\neq\emptyset)$)

Soit $\sigma\in Tr_A$, si $\sigma\in Tr_B$ alors

Puisque $B \leq_{\mathbb{S}} A$ alors d'après Proposition 3.1 et Proposition 3.2 on a

$inp((s_{oA})_\sigma)\subseteq inp((s_{oB})_\sigma) \wedge out((s_{oB})_\sigma)\subseteq out((s_{oA})_\sigma)$

Cas $|\sigma|=0$

Puisque $B \leq_{\mathbb{P}} A^{red}$ alors d'après la définition de A^{red} et Définition 3.12 (i) on a

$$out(s_{0A})\neq\emptyset \Rightarrow out(s_{0B})\neq\emptyset,$$

Cas $|\sigma|>0$

Puisque $B \leq_{\mathbb{P}} A^{red}$ alors d'après la définition de A^{red} et Définition 3.12 (iii) on a

$$out((s_{0A})_\sigma)\neq\emptyset \Rightarrow out((s_{0B})_\sigma)\neq\emptyset,$$

Nous concluons que : $B \leq_{conf} A^{red} \Rightarrow B \leq_{red} A$. □

3.2.3 Transformations pour les automates à entrées et sorties

Si l'on s'intéresse seulement aux traces du système, les classes des automates à entrées et sorties déterministes et non-déterministes sont équivalentes, c'est à dire, pour tout automate à entrées et sorties non-déterministe A on peut déterminer un automate à entrées et sorties déterministe B tels que $Tr_A=Tr_B$. Cette construction est similaire à celle décrite pour les systèmes de transitions étiquetées. À partir de l'automate à entrées et sorties A on dérive le système de transitions étiquetées correspondant E. L'automate à entrées et sorties B est obtenu à partir de l'automate de traces $TM(E)=(S_t, L_t, T_t, s_{to})$ en posant $S_B=S_t$, $I_B=I_A$, $O_B=O_A$, $T_A=T_t$ et $s_{0B}=s_{to}$. De manière analogue, à partir d'un automate à entrées et sorties déterministe, on peut construire un automate à entrées et sorties déterministe ayant le même ensemble de traces et le nombre minimal d'états.

3.3 Modèle des machines de Mealy

3.3.1 La composition

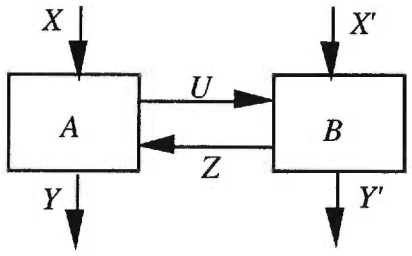


Figure 3.9 : L'architecture de composition de deux machines de Mealy A et B

Nous considérons la classe de systèmes qui peuvent être décrit comme la composition de deux machines de Mealy qui communiquent entre elles et avec l'environnement, telle que illustrée dans la Figure 3.9. Les ensembles d'actions X, X', U, Z, Y and Y' sont considérés deux à deux disjoints. Les actions dans $X \cup X'$ sont contrôlées par l'environnement et seules

les actions dans YUY' sont observées par l'environnement. Nos assumons de plus que chacun des ensembles XUX', YUY', XUZ, UUY, UUX', et ZUY' est non vide.

Le système obtenu par la composition des deux machines de Mealy A et B, ne peut lui même être décrit par une machine de Mealy que si l'environnement respecte une contrainte sur l'ordre des entrées et des sorties. Cette contrainte impose qu'une entrée externe $x \in XUX'$ ne peut être soumise au système qu'après que ce dernier ait produit une sortie $y \in YUY'$ en réponse à l'entrée externe précédante. Formellement, un tel environnement peut être modélisé par le système de transitions étiquetées E_{Env} illustré dans la Figure 3.10, dans le cas où $X=\{x_1, x_2\}$, $Y=\{y_1, y_2\}$, $X'=\{x'_1, x'_2\}$ et $Y'=\{y'_1, y'_2\}$.

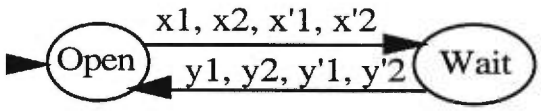


Figure 3.10 : Système de transitions étiquetées E_{Env} associé à un environnement *Env*

Le comportement d'un système formé par deux machines de Mealy communicantes A et B dans un tel environnement, peut être décrit grâce a un système de transitions étiquetées, noté $LTS(A, B)$, et une machine de Mealy, noté $A \Diamond B$ (lorsqu'elle existe). Le système de transitions étiquetées décrit le comportement global du système incluant les actions internes, tandis que la machine de Mealy $A \Diamond B$ décrit le comportement observable du système en fonction seulement des entrées et sorties externes.

Le système de transitions étiquetées $LTS(A, B)$, qui représente le comportement global du système dans l'environnement *Env*, est par définition égal à la composition des systèmes de transitions étiquetées E_A -système de transition étiquetées correspondant à la machine de Mealy A-, E_B -système de transition étiquetées correspondant à la machine de Mealy B- et E_{Env} -système de transition étiquetées correspondant à l'environnement *Env*-, c'est à dire $LTS(A, B)=E_A||E_B||E_{Env}$. Le système composé accepte une entrée externe seulement lorsqu'il est dans un état stable, c'est à dire lorsque le système de transition étiquetées E_{Env} est dans l'état *Open*.

Exemple :

Dans l'exemple suivant, nous illustrons la construction du système de transitions étiquetées $LTS(A, B)$ (Figure 3.13). Les machines de Mealy A et B ainsi que les systèmes de transition étiquetées qui leurs correspondent sont décrits dans les figures 3.11 et 3.12.

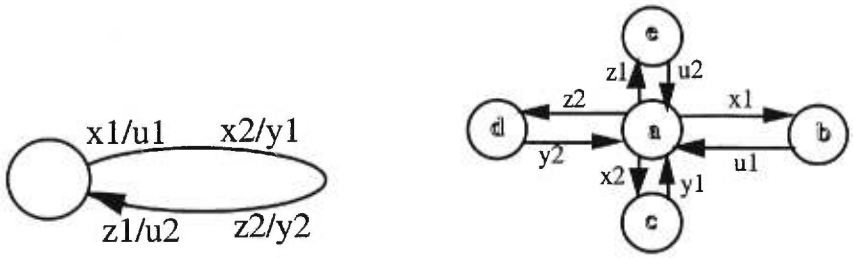


Figure 3.11 : La machine de Mealy *A* ainsi que le système de transition étiquetées E_A

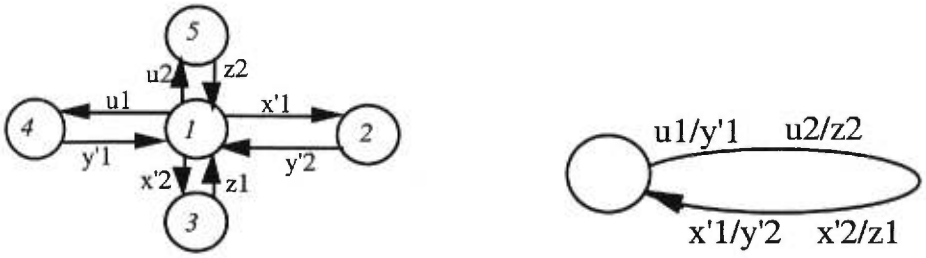


Figure 3.12 : La machine de Mealy *B* ainsi que le système de transition étiquetées E_B

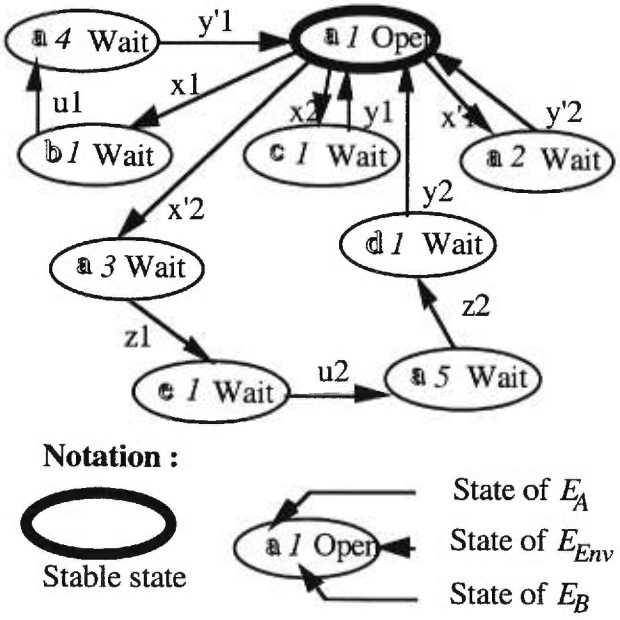


Figure 3.13 : Le système de transitions étiquetées *LTS*(*A*, *B*)

Dans le cas où des cycles étiquetés seulement par des actions internes sont présents dans le système de transitions étiquetées *LTS*(*A*, *B*), le comportement observable du système ne peut pas être représenté par une machine de Mealy. Dans le cas contraire, la machine de Mealy $A \Diamond B$ qui décrit le comportement observable du système en fonction seulement des entrées et sorties externes peut être déduite. Toutes les actions internes sont remplacées par

l'action non observable dans *LTS*(*A*, *B*), puis il est déterminisé. À partir de ce dernier, la machine de Mealy $A \Diamond B$ est obtenue en regroupant chaque paire de transitions (s_1, x, s_2) et (s_2, y, s_3) en une transition s_1- x/y ->s_3. La machine de Mealy $A \Diamond B$ obtenue à partir du système de transitions étiquetées *LTS*(*A*, *B*) de la Figure 3.13 a un seul état. Elle est illustrée dans la figure 3.14.

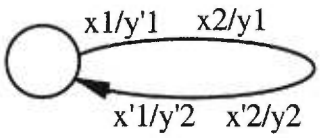


Figure 3.14 : La machine de Mealy $A \Diamond B$

3.3.2 Les relations de conformité

3.3.2.1 Équivalence

Deux états s et t de la machine de Mealy $A=(S, X, Y, h, s_0)$ sont dit équivalents, noté $s \cong t$, si et seulement si :

$$X_s^* = X_t^* \text{ et } \forall \alpha \in X_s^* \ (h^2(s, \alpha) = h^2(t, \alpha)).$$

Une machine de Mealy dont les états sont deux à deux non équivalents est dite minimale.

Les machines de Mealy A et B sont équivalentes si leurs états initiaux sont équivalents, sinon, elles sont non équivalentes.

La relation d'équivalence est parfois appelée équivalence de traces. Des machines équivalentes se comportent de façon identique, i.e. elles exécutent les mêmes traces.

La dérivation de tests de conformité [Chow 78][Ural 87][Sidh 89] à partir d'une spécification décrite par une machine de Mealy déterministe complètement spécifiées (CDFSM) est basée sur la relation d'équivalence. Si A est une spécification CDFSM et B est une implantation CDFSM équivalente à A, alors cette implantation est conforme à la spécification et "$\cong$" est la relation de conformité la plus fine qu'on peut avoir pour le modèle des machines déterministes complètement spécifiées. Chaque implantation CDFSM qui est non équivalente à la spécification CDFSM représente une implantation non conforme.

3.3.2.2 Quasi-équivalence

Étant donnés deux états s et t de la machine de Mealy $A=(S, X, Y, h, s_0)$, l'état s est quasi-équivalent à l'état t, noté $s \leq_{qe} t$, si et seulement si :

$$X_t^* \subseteq X_s^* \text{ et } \forall \alpha \in X_t^* \ (h^2(s, \alpha) = h^2(t, \alpha)).$$

La machine de Mealy B est quasi-équivalente à la machine de Mealy A, noté $B \leq_{qe} A$, si l'état initial de B est quasi-équivalent à l'état initial de A, sinon, B n'est pas quasi-équivalente à A, noté $B \not\leq_{qe} A$.

Cette relation de conformité exprime le faite que toutes les séquences de sortie décrites par la machine de Mealy A et seulement celles-ci doivent être produites par la machine de Mealy B en réponse à toutes les séquences d'entrées acceptées par la machine de Mealy A. La notation "$s \cong_Q t$ " est aussi utilisée pour cette relation.

3.3.2.3 Réduction

Étant donnés deux états s et t de la machine de Mealy $A=(S, X, Y, h, s_0)$. L'état s est une réduction de l'état t, on note $s \leq_{red} t$, si et seulement si :

$$X_t^* \subseteq X_s^* \text{ et } \forall \alpha \in X_t^* \ (h^2(s, \alpha) \subseteq h^2(t, \alpha)).$$

La machine de Mealy B est une réduction de la machine de Mealy A, noté $B \leq_{red} A$, si l'état initial de B est une réduction de l'état initial de A, sinon, B n'est pas une réduction de A, noté $B \not\leq_{red} A$. Si B est déterministe et $B \leq_{red} A$, alors B est dite une *réduction* déterministe de A.

Cette relation de conformité est basée sur le fait que toutes les séquences de sorties produites par la machine de Mealy B en réponse à toutes les séquences d'entrées acceptées par la machine de Mealy A doivent être décrites par la machine de Mealy A. La notation "$s \leq t$ " est aussi utilisée pour cette relation.

Si A et B sont deux machines de Mealy alors :

$$(A \leq_{red} B \text{ et } B \leq_{red} A \Leftrightarrow B \cong A) \text{ et } (A \leq_{qe} B \text{ et } B \leq_{qe} A \Leftrightarrow B \cong A).$$

La relation de quasi-équivalence est un cas particulier de la relation de réduction. Si les machines sont déterministes alors les deux relations se confondent dans la relation de quasi-équivalence pour les machines partiellement spécifiées ou dans la relation d'équivalence pour les machines complètement spécifiées.

3.3.3 Transformations pour les machines de Mealy

Pour les machines de Mealy, la classe des machines non-déterministes non-observables et la classe des machines non-déterministes observables sont identiques au sens des traces. À partir de toute machine de Mealy non-déterministe non-observable A, on peut construire une machine de Mealy non-déterministe observable B ayant le même ensemble de traces.

Une telle construction peut être réalisée de la manière suivante. À partir d'une machine de Mealy non-déterministe non-observable $A=(S, X, Y, h, s_0)$, on construit le système de transitions étiquetées $E=(S, L, T, s_0)$ où :

- $L=X.Y$, un élément de L est obtenu à partir de la concaténation d'un élément de X et d'un élément de Y,
- $(s, x.y, p)\in T$ si et seulement si $(s, x)\in D_A$ et $(p, y)\in h(s, x)$.

Exemple :

Nous illustrons dans la Figure 3.15 la construction d'une machine de Mealy observable à partir d'une machine de Mealy non-observable.

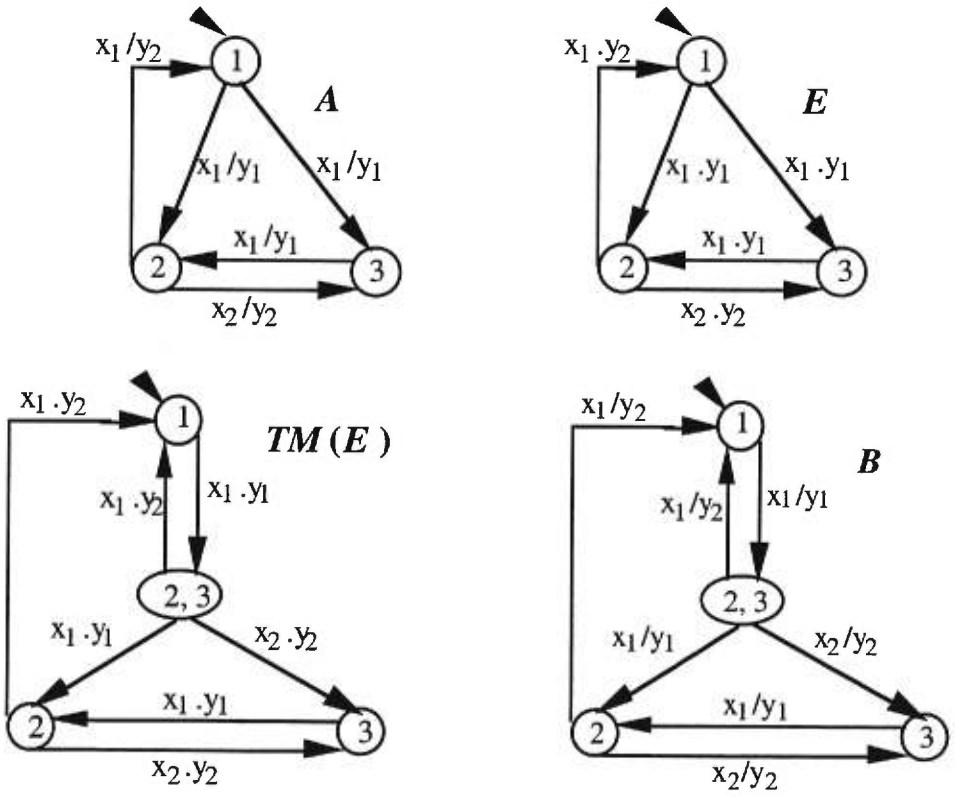


Figure 3.15 : Construction d'une machine de Mealy observable B à partir d'une machine de Mealy non-observable A

La machine de Mealy B sera obtenu à partir de l'automate de traces $TM(E)$ en remplaçant tout simplement chaque étiquette $x.y$ d'une transition par l'étiquette x/y.

De même, à partir de toute machine de Mealy observable A, on peut construire une machine de Mealy B minimal ayant le même ensemble de traces. La construction est similaire à celle décrite précédemment à part que la machine B sera obtenu à partir du système de transitions étiquetées minimal associé à E.

CHAPITRE 4

LA CONSTRUCTION DANS LE MODÈLE DES SYSTÈMES À TRANSITIONS ÉTIQUETÉES

Dans ce chapitre, nous décrivons des travaux existants reliés à la construction de sous-modules ainsi qu'à la synthèse de contrôleurs. La plupart des travaux sur la construction de sous-modules utilisent comme modèle pour décrire les spécifications celui des systèmes à transitions étiquetées. Par contre, les travaux sur la synthèse de contrôleurs utilisent la théorie des langages ainsi que les automates temporisés.

4.1 Les travaux sur la construction de sous-modules

4.1.1 La méthode Merlin-Bochman

Merlin et Bochmann ont proposé dans [Merlin 83] une première solution pour le problème de construction de sous-module. Les spécifications sont données sous la forme d'ensembles de séquences d'exécution (traces) et la relation d'inclusion de traces est utilisée comme relation de conformité. Le principe de l'approche est le suivant : si le système comprend n modules, que sa spécification ainsi que la spécification de (n-1) modules sont données, alors la méthode permet de déterminer, à l'aide d'une formule, la spécification du $n^{\text{ième}}$ module. Chaque module M_i est caractérisé par un ensemble V_i (alphabet du module M_i) qui contient les actions observables de l'extérieur, qui peuvent être exécutées par le module. Lorsqu'un module désire entreprendre une action conjointe avec un autre module (interaction) la communication entre les deux modules se fait par rendez-vous.

Pour décrire de façon simple la méthode, nous considérons le cas d'un système S composé de deux modules M_1 et M_2. La spécification du système S est donnée par l'ensemble de traces Tr_S sur l'alphabet V_S. La spécification du module connu M_1 est donnée

par l'ensemble de traces Tr_1 sur l'alphabet V_1. L'alphabet du module inconnu M_2, si il existe, est obtenu grâce à la formule suivante :

$V_2=(V_s-V_1)\cup(V_1-V_s)$ où l'opérateur "-" désigne la différence ensembliste.

La spécification du module inconnu est obtenu, si elle existe, grâce à la formule :

$$Tr_1=Pr_{V_2}(Tr_s\times Tr_1)-Pr_{V_2}(\neg Tr_s\times Tr_1),$$

l'expression $(Tr_s\times Tr_1)$ représente le produit cartésien des deux spécifications. Cependant, en accord avec le mode de communication, les actions communes aux deux modules ne peuvent être entreprises que si les deux modules se trouvent chacun dans un état leur permettant de le faire. L'ensemble de traces "$\neg Tr_s$" représente le complément de Tr_s dans V_s*, c'est l'ensemble des séquences d'exécution formées par des actions dans V_s non présentes dans Tr_s. L'opérateur "Pr_{V_2}" représente la projection d'une spécification sur l'alphabet V_2 du module M_2.

La solution obtenue par cette méthode décrit le plus grand ensemble de traces possibles pour le module . C'est à dire, $Pr_{V_s}(Tr_1\times Tr_2)\subseteq Tr_s$ et pour toute solution possible Tr on a $Tr\subseteq Tr_2$. Cependant, la relation de conformité utilisée ne permet pas de détecter les blocages. De plus, lorsque on considère des ensembles de traces qui sont fermés par rapport à l'opérateur préfixe pour les spécifications du système et du module connu, la méthode ne produit pas nécessairement une solution dont l'ensemble des traces est fermé par rapport à l'opérateur préfixe. Un exemple illustrant ce cas est décrit dans [Kelekar 93].

Une implantation de la méthode se trouve dans le travail [Sidhu 89]. Une reformulation de la même méthode se trouve dans [Hagheverdi 96]. Une extension de la méthode se trouve dans le travail [Kelekar 94]. Les auteurs utilisent un modèle asymétrique qui permet de distinguer entre les actions reçues et celles émises par un module afin de représenter la causalité dans les communications. Un avantage de cette représentation réside dans la possibilité de détecter les réceptions non spécifiées. Des propriétés de progrès sont aussi prises en compte lors de la génération de la solution.

4.1.2 La méthode de Parrow

Dans son travail [Parrow 89], Parrow présente une méthode pour la résolution du problème de construction de sous-module. Le modèle utilisé pour les spécifications est CCS "Calculus of Communicating Systems" [Milner 80] et l'équivalence observationnelle est choisie pour la relation de conformité. Dans CCS, une spécification est appelée un agent. Le problème est formulé sous la forme de l'équation $(A|X)\backslash_L\approx S$, où X est l'inconnu, S représente le système entier, A représente le sous-module connu et L décrit les canaux de

communications à travers lesquels les modules interagissent.

La méthode, appelée méthode tableau, est basé sur des transformations successives d'équations en des équations plus simples en parallèle afin de générer une solution. Un tableau est composé de deux parties : un but B et un environnement E. Un environnement est une application partielle d'un ensemble d'identificateurs vers l'ensemble des agents. Un identificateur est lié par un environnement si il appartient au domaine de celui-ci, sinon il est libre. L'idée derrière l'utilisation d'un tableau est la représentation d'une étape intermédiaire dans la production d'une solution. Le but indique ce qui reste à satisfaire, alors que l'environnement enregistre la solution obtenue à cette étape. À l'étape initiale, le tableau contient le but $(A|X)\backslash_L \approx S$ et un environnement où A et S sont liés et X est libre, ce qui signifie qu'il reste à trouver une extension de l'environnement qui satisfait le but. La méthode est basée sur des règles de transformation permettant de passer d'un tableau représentant une étape donnée à un autre tableau représentant l'étape suivante. Il y a deux types de règles :

- Instantiation : cette règle permet d'étendre l'environnement en estimant les transitions initiales de l'agent inconnu par l'utilisation du théorème d'expansion. Si (B, E) est un tableau et X est libre, alors $X \mapsto \sum_{i=1}^{n} a_i X_i$ peuvent être ajoutées à E. Les a_i représentent des actions et les X_i représentent des identificateurs libres et distincts. L'ensemble $\{a_1, ..., a_i, a_n\}$ décrit les actions initiales de X. En général, il y a plusieurs tels ensembles correspondant à différentes solutions. Si l'instantiation est appliquée sans soin, ceci peut mener à un tableau impossible à satisfaire. Afin de résoudre ce problème, Parrow introduit certaines heuristiques pour le choix de cet ensemble.
- Réduction : cette règle permet de simplifier le but. Il y a deux cas :
  - Équivalence : une équation $C \approx D$, où C et D sont observationnellement équivalent dans l'environnement E, peut être éliminée. Si c'est la seule équation dans le but, alors celui-ci devient vrai.
  - Décomposition : si le but contient une équation $C \approx D$, avec $C \mapsto \sum_{i=1}^{n} a_i C_i$, et $D \mapsto \sum_{i=1}^{n} a_i D_i$ pour $1 \leq i \leq n$, alors cette équation peut être remplacée par $\bigwedge_{i=1}^{n} (C_i \approx D_i)$.

Ces transformations sont appliquées tant que l'on peut progresser. Lorsque la méthode se termine deux cas peuvent se présenter. On a atteint un tableau impossible à satisfaire, dans ce cas soit il n'existe pas de solution, soit l'instantiation a été mal dirigée durant un étape antérieure. Le but est égale à vrai, dans ce cas X est lié dans l'environnement,

c'est une solution. Cependant la solution construite par la méthode peut ne pas être la plus générale à cause de l'application des heuristiques dans la règle d'instantiation. La méthode se limite au cas ou la spécification du système désiré est déterministe.

4.1.3 La méthode de Qin-Lewis

Dans leur travail [Qin 91], Qin et Lewis présentent une autre méthode pour la résolution du problème de construction d'un sous-module. Pour décrire les spécifications, ils utilisent le modèle des systèmes à transitions étiquetées. Comme relations de conformité, ils considèrent dans un premier temps l'équivalence observationnelle puis dans un deuxième temps la bissimulation forte.

Dans le cas où la relation de conformité est la bissimulation forte, l'algorithme pour l'obtention de la solution R de l'équation $P||X \sim Q$, est composé des cinq étapes suivantes :

Étape 1 : On pose A_P=Alphabet(P), A_Q=Alphabet(Q), $Ex(R)=A_Q-A_P$, $Com(R)=\{\overline{\alpha} \mid \alpha \in A_P-A_Q\}$, $A_R=Ex(R)\cup Com(R)$ et $Ex(P)=A_P-(\widehat{A}_P\cap\widehat{A}_R)$ où $\widehat{A}=A\cup\{\overline{\alpha} \mid \alpha \in A\}$. Générer toutes les paires (p, q) pour tout état p dans P et tout état q dans Q. Marquer la paire (p, q) "mauvaise" lorsque soit il existe une transition à partir de p, étiquetée avec une action u dans $Ex(P)\cup\{\tau\}$, mais qu'il n'existe pas de transition étiquetée par u à partir de q, ou soit il existe une transition à partir de q, étiquetée avec une action u dans $Ex(P)$, mais qu'il n'existe pas de transition étiquetée par u à partir de p.

Étape 2 : Si $p \xrightarrow{u} p'$ dans P et $q \xrightarrow{u} q'$ dans Q, on note $(p, q) \xrightarrow{u}_{I_s} (p', q')$. De plus, si pour au moins une action u on a $(p, q) \xrightarrow{u}_{I_s} (p', q')$, on note $(p, q) \rightarrow_{I_s} (p', q')$. On note $I_s(p, q)=\{(p', q') \mid (p, q) \Rightarrow_{I_s} (p', q')\}$ où $\Rightarrow_{I_s}$ est la fermeture réflexive et transitive de $\rightarrow_{I_s}$. Créer un boite (état) initiale X_0 contenant les paires d'états dans $I_s(p_0, q_0)$ où p_0 représente l'état initial de P et q_0 représente l'état initial de Q. Marquer la boite X_0 "non traitée". Associer à chaque boite un ensemble de ports étiquetés avec les actions dans $A_R\cup\{\tau\}$.

Étape 3 : Faire ce qui suit tant que il existe une boite X_i "non traitée" :

a - Si il existe dans X_i une paire (p, q) marquée "mauvaise" alors marquer la boite X_i "mauvaise" et "traitée", sinon faire les traitements dans b, c et d puis marquer la boite X_i "traitée".

b - pour toute paire $(p, q) \in X_i$, créer une arête étiquetée par u reliant (p, q) à (p', q') chaque fois qu'on a $(p, q) \xrightarrow{u}_{I_s} (p', q')$.

c - Faire pour toute action u dans $Ex(R)\cup\{\tau\}$ ce qui suit :

si il existe $(p, q)\in X_i$ telle que il n'existe pas de transition étiquetée par u à partir de q, alors marquer le port u dans la boite X_i "mauvais",

sinon poser $X_i(u)=\{I_s(p, q')|\ (p, q)\in X_i$ et $q \xrightarrow{u} q'$ dans $Q\}$, par la suite si il n'existe aucune boite X_j contenant exactement les éléments dans $X_i(u)$, créer une telle boite et la marquer "non traitée", enfin créer une arête étiquetée par u reliant (p, q) dans X_i à (p, q') dans X_j et une transition $X_i \xrightarrow{u} X_j$.

d - Faire pour toute action $\overline{\alpha}$ dans $Com(R)$ ce qui suit :

si pour toute paire $(p, q)\in X_i$ il n'existe pas de transition étiquetée par α à partir de p alors créer la transition $X_i \xrightarrow{\overline{\alpha}}$ DON'T_CARE où DON'T_CARE est une boite spéciale.

si il existe une paire $(p, q)\in X_i$ telle que il existe une transition à partir de p, étiquetée avec α, mais il n'existe pas de transition étiquetée par τ à partir de q, alors marquer le port α "mauvais".

Si aucun des deux cas précédant n'est vrai, poser $X_i(\alpha)=\{I_s(p', q')|\ (p, q)\in X_i$, $p \xrightarrow{\alpha} p'$ dans P et $q \xrightarrow{\tau} q'$ dans $Q\}$, par la suite si il n'existe aucune boite X_j contenant exactement les éléments dans $X_i(\alpha)$, créer une telle boite et la marquer "non traitée", enfin créer une arête étiquetée par u reliant (p, q) dans X_i à (p, q') dans X_j chaque fois que $p \xrightarrow{\alpha} p'$ dans P et $q \xrightarrow{\tau} q'$ et une transition $X_i \xrightarrow{\overline{\alpha}} X_j$.

Étape 4 : Répéter tant que il existe des boites marquée "mauvaise" :

a - si il existe une transition étiquetée par une action u dans $A_R\cup\{\tau\}$ à partir d'une boite X_i vers une boite X_j marquée "mauvaise", alors marquer le port u dans X_i "mauvais", puis éliminer toutes les transitions à partir de X_i étiquetées par u et éliminer toutes les arêtes reliant des paires d'états dans X_i à des paires d'états dans X_j.

b - marquer la boite X_i "mauvaise" si il existe une paire $(p, q)\in X_i$ telle que :

i - $q \xrightarrow{u} q'$ dans Q avec $u\neq\tau$ mais il n'existe pas d'arête telle que $(p, q) \xrightarrow{u} (p', q')$, ou

ii - $q \xrightarrow{\tau} q'$ dans Q mais il n'existe pas d'arête telle que $(p, q) \xrightarrow{\tau} (p', q')$ ou

$(p, q) \xrightarrow{\overline{\alpha}} (p', q')$ avec $\overline{\alpha} \in Com(R)$.

Étape 5 : Si la boite initiale est marquée "mauvaise" alors in n'existe pas de solution sinon le système de transitions étiqueté $R=(S_R \cup \{\text{DON'T\_CARE}\}, A_R \cup \{\tau\}, T_R, X_0)$ où S_R est l'ensemble contenant les boites "bonnes" qui restent et T_R est l'ensemble contenant les transitions à partir des éléments dans S_R vers des éléments dans $S_R \cup \{\text{DON'T\_CARE}\}$.

Dans cette algorithme, une boite sert à représenter un état de la solution. Dans l'étape 1, une paire (p, q) est marquée "mauvaise" si il est impossible d'avoir un état r d'une solution tel que $(p, r) \sim q$. Dans l'étape 2, on initialise la construction de la solution en plaçant dans la première boite l'ensemble $I_s(p_0, q_0)$ où p_0 représente l'état initial de P et q_0 représente l'état initial de Q. Intuitivement, l'ensemble $I_s(p, q)$ décrit l'état q' auquel l'état q devrait se rendre avec une certaine séquence σ non observable par la solution lorsque σ mène de p à p' afin d'avoir $(p, r) \sim q$. Dans l'étape 3, l'algorithme construit une machine en permettant toutes les transitions possibles dans le but d'obtenir la solution la plus générale. Dans l'étape 4, on élimine toutes les "mauvaises" boites, c'est-à-dire, les boites dont la présence causera l'apparition d'un comportement non conforme. Finalement, dans l'étape 5 on construit effectivement une solution lorsqu'elle existe.

Comparée à la méthode de Parrow, cette méthode est entièrement automatique. Elle ne requiert pas la participation du concepteur. Mais elle se limite aussi au cas où la spécification du système désiré est déterministe.

4.1.4 D'autres travaux

Dans [Shields 89], l'auteur utilise le modèle CCS "Calculus of Communicating Systems" [Milner 80] pour décrire les spécifications. Il considère deux modules P_1 et P_2 qui interagissent à travers une interface inconnue X et doivent satisfaire une spécification donnée Q. Ceci peut être formaliser par l'équation $(P|X)\backslash_L \approx Q$, où $P=P_1|P_2$, X et Q sont des termes CCS, | est l'opérateur de composition parallèle, $\backslash_L$ est l'opérateur de restriction pour les actions de synchronisation et $\approx$ est l'équivalence observationnelle. Cette équation est appelée équation d'interface. Il donne les conditions nécessaires et suffisantes pour l'obtention d'une solution dans le cas où P et Q sont des processus ayant un nombre fini d'états et Q est déterministe.

Dans la thèse [Tao 96], l'auteur propose une méthode pour la construction de sous-

modules qui généralise les travaux précédants en tenant en compte le fait que les événements observables ne sont pas nécessairement contrôlables. Pour décrire les spécifications, il utilise le modèle des systèmes à transitions étiquetées. La relation de conformité est le préordre de test. Cette méthode traite le non déterminisme en transformant la spécification du service désiré en un graphe de refus [Brinksma 88]. Un concept appelé Machine avec États Groupés (MEG) est utilisé pour trouver une solution maximale. La méthode est par la suite généralisé au cas des systèmes temporisés dans le cas où le système non temporisé représentant la spécification du système désiré est déterministe et lorsque la contrainte temporel associée à chaque transition est reliée à au plus un temporisateur.

4.2 Les travaux sur la synthèse de contrôleurs

4.2.1 Les travaux de Ramadge et Wonham

La synthèse de contrôleurs pour les systèmes à événements discrets a été largement étudiée dans les travaux de Ramadge et Wonham [Ramadge 89]. Dans leur approche, le système à événements discrets à contrôler est modélisé par un système de transitions étiquetées et son comportement par un langage formel. De plus, certains états sont considérés comme des balises (marker state), il servent à signaler que certaines tâches ont été complétées. Le système à événements discrets est considéré comme un équipement qui effectue des transitions d'états et qui génère des séquences d'événements. La caractéristique du contrôle réside dans le fait que certains événements (transitions) peuvent être empêchés par un contrôleur externe. L'idée est de construire un contrôleur qui forcera le langage généré par le système à événements discrets à satisfaire une inclusion dans un langage donné. Afin de modéliser l'ensemble des événements contrôlables, l'alphabet du langage est partionné en deux sous ensembles, le premier représente l'ensemble des événements contrôlables, c'est-à-dire ceux qui peuvent être empêchés par le contrôleur, alors que le second représente l'ensemble des événements non contrôlables. Les premiers résultats ont caractérisés les langages dit contrôlable.

Définition 4.1 : Pour un système à événements discrets G générant le langage $L(G)$ et ayant Σ_u pour ensemble d'événements non contrôlables, un sous langage $K \subseteq L(G)$ est contrôlable si $\underline{K}\Sigma_u \cap L(G) \subseteq \underline{K}$, où $\underline{K}\Sigma_u$ représente l'ensemble des éléments obtenus à partir de la concaténation d'un élément dans $\underline{K}$ avec un élément dans Σ_u et $\underline{K}$ représente la fermeture de K par l'opérateur préfixe.

Cette définition décrit une condition nécessaire que doit vérifier tout sous ensemble du

langage généré par un système à événements discrets afin qu'il soit contrôlable. La propriété impose que l'extension de toute séquence d'événements qui est préfixe d'un élément du sous langage contrôlable avec un événement non contrôlable doit aussi être préfixe d'un élément du sous langage puisqu'on ne peut pas empêcher un tel événement.

Lorsqu'on fait jouer aux états balises le rôle d'états acceptants, le langage associé, appelé langage marqué et noté $L_m(G)$, est défini par :

$L_m(G)=\{\sigma|\ \sigma$ est une séquence d'événements menant de l'état initiale à un état balise$\}$, on note que $L_m(G)\subseteq L(G)$. Si $\underline{L_m}(G)=L(G)$ alors le système est dit non bloquant.

Pour un système à événements discrets non bloquant générant le langage $L(G)$ et ayant pour langage marqué $L_m(G)$, les résultats suivants sont obtenus [Ramadge 89]:

Résultat 1 - Pour un sous ensemble non vide $K\subseteq L(G)$, il existe un contrôleur C tel que le langage généré par le système sous la supervision du contrôleur, noté $L(G, C)$, est égale à K si et seulement si K est préfixe fermé et contrôlable.

Résultat 2 - Pour un sous ensemble non vide $K\subseteq L_m(G)$, il existe un contrôleur C tel que le système sous la supervision du contrôleur est non bloquant et son langage marqué, noté $L_m(G, C)$, est égale à K si et seulement si K est contrôlable et $\underline{K}\cap L_m(G)= K$.

Résultat 3 - Si le sous ensemble K ne satisfait ni les conditions du Résultat 1, ni celles du Résultat 2, alors il existe un sous ensemble maximal unique de K qui est contrôlable.

Basé sur les résultats précédants, un algorithme a été développé pour déterminer le sous ensemble maximal unique de K qui est contrôlable. Il utilise la technique du point fixe [Tarski 55].

4.2.2 Les travaux de Thistle et Wonham

Dans leur travail [Thistle 94], Thistle et Wonham étendent les travaux sur la synthèse de contrôleurs au cas où les langages sont représentés par des automates ayant un nombre fini d'états et décrivent des séquences infinies d'événements (ω-langages). Dans le cas des langages décrivant des séquences finies d'événements, l'inclusion de langage permet de spécifier des propriétés de sécurité, c'est-à-dire, que certaines séquences d'événements ne seront jamais produites par le système. Par contre, les propriétés de vivacité qui requièrent que certaines séquences désirables peuvent éventuellement être obtenues ne peuvent pas être exprimées grâce à l'inclusion de langages. En effet, les propriétés de sécurité ont été

formellement définies dans [Alpern 85] comme représentant des restrictions sur les séquences finies d'événements, alors que les propriétés de vivacité ont été définies comme n'imposant aucune restriction sur les séquences finies d'événements mais plutôt comme représentant des restrictions sur les séquences infinies d'événements.

Une propriété appropriée - ω-contrôlabilité- est définie. Par la suite, une caractérisation du sous ensemble de contrôlabilité basée sur l'approche du calcul du point fixe de Emerson et Jutla [Emerson 88] est décrite. Le sous ensemble de contrôlabilité est défini comme l'ensemble des états à partir desquels le ω-automate peut être contrôlé afin de produire seulement les séquences d'événements qui satisfont la condition d'acceptance. Cette condition d'acceptance est basée sur l'ensemble des états visités un nombre infini de fois durant la génération de la séquence infinie d'événements. Le sous ensemble de contrôlabilité est construit dans le cas d'un automate ayant un nombre fini d'états et équipé de deux condition de Rabin, la première représentant une spécification et la deuxième représentant une hypothèse de modélisation qui peut par exemple capturer une propriété d'équité. Le sous ensemble de contrôlabilité est défini dans [Thistle 95], comme l'ensemble des états à partir des quels le ω-automate peut être contrôlé afin que toute séquence infinie d'événements qui est consistante avec les actions de contrôle et satisfait l'hypothèse de modélisation satisfera aussi la spécification.

4.2.3 La synthèse de contrôleurs temporisés

Les travaux sur la synthèse de contrôleurs ont été étendu au cas des systèmes à événements discrets temps-réel dans [Maler 95]. Les auteurs s'intéressent aux systèmes temps-réel, ils modélisent de tels systèmes par des automates temporisés [Alur 94], c'est-à-dire, des automates équipés avec des horloges dont les valeurs augmentent continûment dans le temps lorsque l'automate est dans n'importe lequel de ses états. Les valeurs des horloges peuvent interférer avec les transitions en apparaissant dans des contraintes temporelles associées aux transitions. Une transition ne pourra par exemple être exécutée qu'à des moments précis dans le temps. Une transition peut aussi remettre à zéro certaines horloges.

Il est prouvé dans leur travail que le problème de la synthèse de contrôleurs est solvable lorsque la spécification du système à événements discrets est donnée sous la forme d'un automate temporisé. Ceci signifie qu'un automate temporisé pourra être obtenu (lorsque c'est possible) tel que le système sous sa supervision n'exhibera que des comportements acceptables. La solution est obtenue par la résolution d'équations grâce à la technique du point fixe. Dans ce travail, les auteurs supposent que le contrôleur peut observer précisément

la configuration global du système. Cependant, dans des situations réelles la configuration du système ne peut être observé que partiellement, et le contrôleur doit fonctionner avec une certaine incertitude sur l'état du système.

4.3 Conclusion

Les travaux décrits dans ce chapitre se basent principalement sur le modèle des systèmes à transitions étiquetées. Le mode de communication utilisé est celui de la communication par rendez-vous. Ce genre de communication est bloquant dans le sens où si une composante impliquée dans la communication n'est pas dans un état où l'action commune est possible la réalisation de cette action sera bloquée. Dans notre travail, nous avons considéré le modèle des automates à entrées et sorties [Lynch 88]. La communication est réalisée par envoi et réceptions de messages. Dans ce type de communication, une émission d'un message ne peut jamais être refusée. C'est cette absence de refus qui caractérise un modèle de communication par entrées et sorties, à la différence du mode de communication par rendez-vous, où la notion de refus est au contraire très importante. De plus, ceci est plus prêt de la réalité, car ça permet de modéliser des composante autonomes. Chaque composante est libre de produire une sortie étiquetée par une action o sans se soucier si les composantes dont les alphabets des entrées contiennent l'action o sont dans des états où l'action o est présente. De plus, dans le cas où le modèle utilisé pour décrire les spécifications est celui des machines de Mealy, l'application directe des méthodes décrites plus haut dans ce chapitre ne permet pas en général l'obtention d'une solution sous la forme d'une machine de Mealy.

CHAPITRE 5

LA CONSTRUCTION DANS LE MODÈLE DES MACHINES DE MEALY

Dans ce chapitre, nous présentons le premier travail réalisé au cours de cette recherche [Drissi 98]. Il nous a permis de bien comprendre la problématique liée à la construction de sous-modules. Le modèle utilisé pour décrire les spécifications du contexte et du système désiré est celui des machines de Mealy déterministes complètement spécifiées. La relation de conformité est la relation d'équivalence.

5.1 Description du problème

Étant données deux machines de Mealy déterministes et complètement spécifiées, A ayant pour ensemble d'entrées $X \cup X'$ et pour ensemble de sorties $Y \cup Y'$, et C ayant pour ensemble d'entrées $X \cup Z$ et pour ensemble de sorties $Y \cup U$, nous désirons obtenir toutes les machines de Mealy déterministes ayant pour ensemble d'entrées $X' \cup U$ et pour ensemble de sorties $Y' \cup Z$ qui sont solutions de l'équation $C \Diamond Comp \cong A$ où $Comp$ est une variable libre.

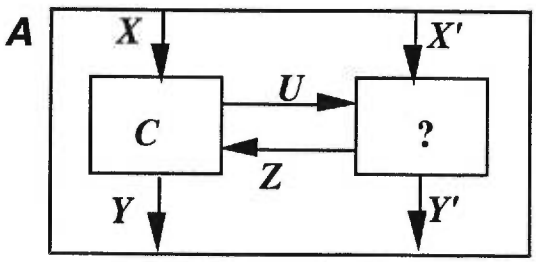


Figure 5.1 : L'architecture du système

Une machine de Mealy déterministe B ayant pour ensemble d'entrées $X' \cup U$ et pour ensemble de sorties $Y' \cup Z$ est une *solution* de l'équation $C \Diamond Comp \cong A$ si et seulement si la

machine de Mealy $C \Diamond B$ existe et est équivalente à A.

Après la réception d'une entrée externe, les modules présents dans le système peuvent avoir un nombre quelconque d'interactions internes avant de produire une sortie externe. Le travail [Watanabe 93a] traite le problème de la construction de sous-modules dans le cas où la communication est synchrone et où les ensembles X' et Y' de la Figure 5.1 sont vides. Il décrit une méthode, basée sur la technique du point fixe [Tarski 55], qui permet de représenter l'ensemble des comportements permis pour le module à concevoir (si il est non vide) sous la forme d'une machine de Mealy non déterministe.

5.2 L'approche de résolution de l'équation

Nous considérons une machine de Mealy chaotique (Figure 5.2), définie par $Ch=(\{ch\}, X' \cup U, Y' \cup Z, H, ch)$, où $H(ch, v)=\{(ch, w) \mid w \in Y' \cup Z\}$ pour tout $v \in X' \cup U$. La machine de Mealy chaotique représente toutes les traces sur l'ensemble d'entrées $X' \cup U$ et l'ensemble de sorties $Y' \cup Z$, elle décrit tous les comportement possibles de la composante à concevoir.

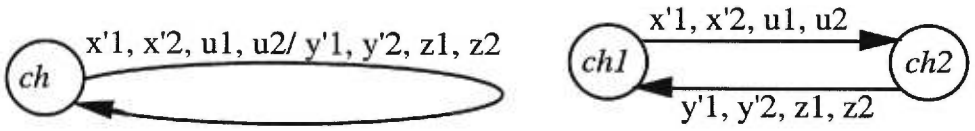


Figure 5.2 : Exemple d'une machine de Mealy chaotique et du système de transitions étiquetées I_{Ch} qui lui correspond

Une trace $\beta \ltimes \delta$ de la machine de Mealy chaotique est interdite à cause d'une séquence d'actions externes $\alpha \in (X \cup X')^*$ si et seulement si pour toute machine de Mealy ayant la séquence d'entrées et sorties β/δ, la composition de cette dernière avec le contexte donné C produit un comportement différent de la machine de Mealy A par rapport à la séquence d'entrées α; sinon la trace $\beta \ltimes \delta$ est permise par rapport à α. Afin de classer les traces de la machine de Mealy chaotique en deux catégories, les traces interdites et les traces permises, nous construisons le système de transitions étiquetées $LTS(C, Ch)=I_C \| I_{Ch} \| I_E$. Puis nous complétons le système de transition étiquetées I_A correspondant à la machine de Mealy A en ajoutant à chaque état contenant une transition sortante étiquetée par un élément $y \in Y \cup Y'$, des transitions étiquetées par chacun des éléments de $(Y \cup Y') \setminus \{y\}$ et menant à un nouvel état silencieux, c'est à dire un état où aucune transition sortante n'est présente. On note $\hat{I}_A$ le système de transition étiquetées ainsi obtenu. Par la suite, nous construisons la composition

de $LTS(C, Ch)$ et de $\hat{I}_A$, et nous notons $I_{A,C}$ le système de transitions étiquetées obtenu. Nous remplaçons dans $I_{A,C}$ les entrées et les sorties externes du contexte C par l'action interne τ, ensuite nous déterminons l'automate de traces correspondant, et on le note $P_{A,C}$. Le système de transitions étiquetées $P_{A,C}$ a pour alphabet l'ensemble $X' \cup U \cup Z \cup Y'$, de plus il caractérise toutes les traces interdites. Une trace interdite $\beta \ltimes \delta$ par rapport à une séquence d'entrées externes α a un préfixe dans $P_{A,C}$ menant à l'état silencieux. L'étape suivante consiste à transformer le système de transitions étiquetées $P_{A,C}$ en une machine de Mealy, notée $[[A,C]]$, où toutes les séquences d'entrées et sorties correspondantes à des traces interdites mènent dans $[[A,C]]$ de l'état initial à un état spécial *FAIL*. La dernière étape consiste à dériver de $[[A,C]]$ une machine de Mealy (si elle existe), notée $[[A,C]]_f$, qui contient seulement les traces permises. Toute solution de l'équation $C \Diamond Comp \cong A$ est une réduction de $[[A,C]]_f$, mais $[[A,C]]_f$ peut avoir des réductions qui ne sont pas des solutions car leurs compositions avec le contexte C ne peuvent être modélisées par une machine de Mealy à cause de cycles étiquetés seulement par des actions internes (livelocks). Toute réduction de $[[A,C]]_f$ est une machine de Mealy ayant pour ensemble d'entrées $X' \cup U$ et pour ensemble de sorties $Y' \cup Z$ ne contenant aucune trace interdite; nous appelons une telle machine une *solution potentielle*.

Si l'ensemble des D-réductions d'une machine de Mealy G coïncide avec l'ensemble des solutions de l'équation $C \Diamond Comp \cong A$ alors G est appelée la *solution générique*.

5.3 Construction de la machine de Mealy $[[A,C]]$

Nous présentons dans ce qui suit une méthode pour la construction du système de transitions étiquetées $[[A,C]]$. Nous illustrons les différentes étapes à travers un exemple. La machine de Mealy A représentant la spécification du système désiré ainsi que le système de transition étiquetées qui lui correspond sont décrits dans la Figure 5.3. La machine de Mealy C représentant la spécification du contexte ainsi que le système de transition étiquetées qui lui correspond sont décrits dans la Figure 5.4.

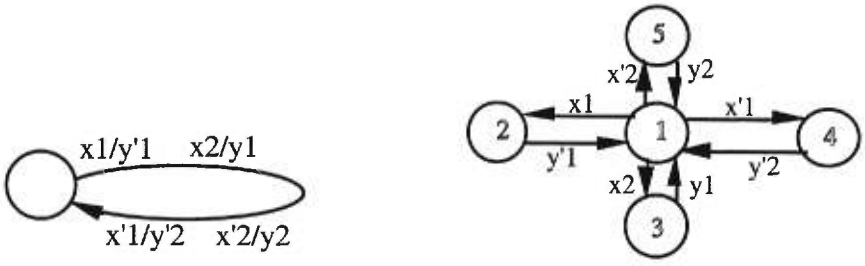


Figure 5.3 : La machine de Mealy A ainsi que le système de transition étiquetées I_A

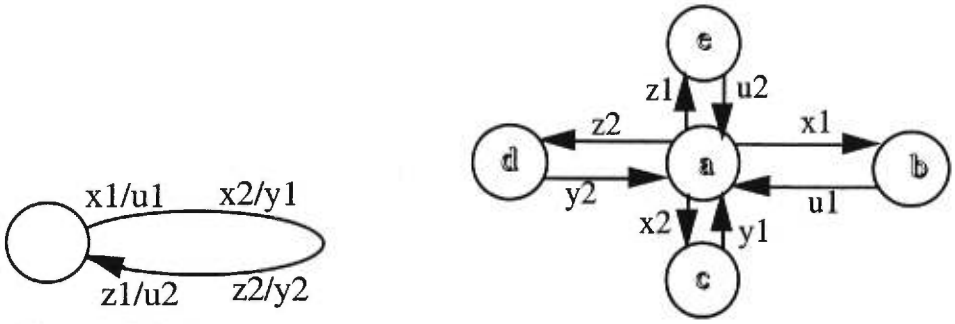


Figure 5.4 : La machine de Mealy C ainsi que le système de transition étiquetées I_C

Étape 1. Construire le système de transitions étiquetées $LTS(C, Ch)=I_C \| I_{Ch} \| I_E$ décrivant le comportement global du contexte et de machine de Mealy chaotique dans l'environnement E (Figure 5.5).

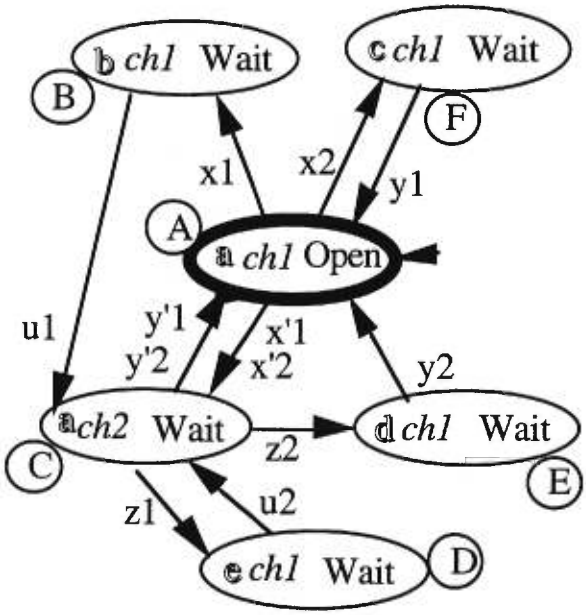


Figure 5.5 : Le système de transitions étiquetées $LTS(C, Ch)=I_C \| I_{Ch} \| I_E$

Étape 2. Pour chaque état du système de transitions étiquetées I_A contenant une transition sortante étiquetée par un élément $y \in Y \cup Y'$, nous ajoutons des transitions étiquetées par chacun des éléments de $(Y \cup Y') \setminus \{y\}$ et menant à un nouvel état silencieux, c'est à dire un état où aucune transition sortante n'est présente, on note $\hat{I}_A$ le système de transition étiquetées obtenu (Figure 5.6). Par la suite, nous construisons la composition de $LTS(C, Ch)$ et $\hat{I}_A$ (Figure 5.7), notée $I_{A,C}$, afin de comparer les traces du système de transition étiquetées $LTS(C, Ch)$ avec celles de $\hat{I}_A$. L'ajout de transitions dans I_A permet de représenter dans $I_{A,C}$ toutes les traces interdites de la composante à concevoir.

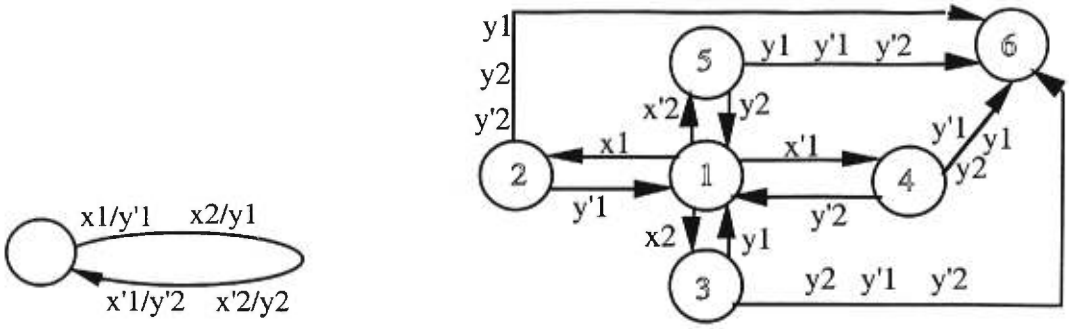


Figure 5.6 : La machine de Mealy A et le système de transitions étiquetées $\hat{I}_A$.

Proposition 5.1 : Étant donnée une trace $\alpha=\alpha_1 y$ sur l'alphabet $X \cup X' \cup U \cup Z \cup Y \cup Y'$, α mène dans $I_{A,C}$ de l'état initial à l'état silencieux si et seulement si α est une trace du système de transitions étiquetées $LTS(C, Ch)$ et $\mathrm{Pr}_{Y \cup Y'}(\alpha_1)y'$, avec $y \neq y'$, est la séquence de sorties produite par la machine de Mealy A en réponse à $\mathrm{Pr}_{X \cup X'}(\alpha)$.

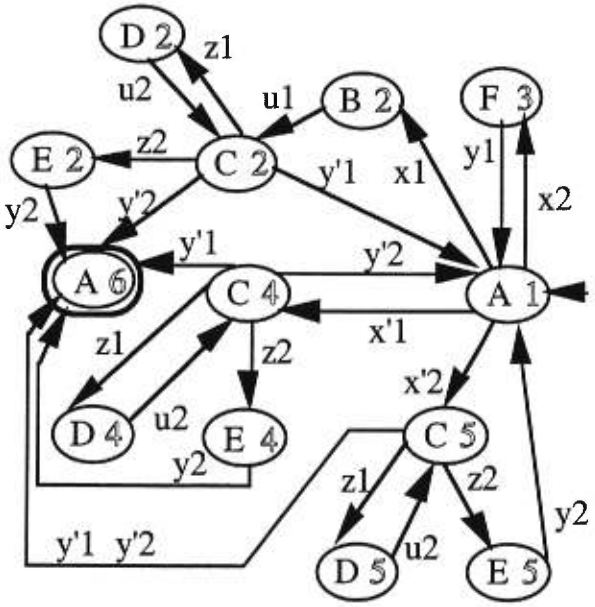


Figure 5.7 : La composition $I_{A,C}$

En d'autres mots, une trace du système de transitions étiquetées $I_{A,C}$, menant dans $I_{A,C}$ de l'état initial à l'état silencieux, représente une trace du système de transitions étiquetées $LTS(C, Ch)$ dont la projection sur l'alphabet de A n'est pas une trace de A. Nous signalons à ce niveau que la Proposition 5.1 est vraie seulement pour des machines de Mealy déterministes A et C car nous supposons que chaque machine produit exactement une séquence de sorties pour toute séquence d'entrées.

Étape 3. Dans cette étape, nous dérivons à partir de $I_{A,C}$ un système de transition étiquetées représentant la projection des traces de $I_{A,C}$ sur l'alphabet $X' \cup U \cup Y' \cup Z$. L'automate obtenu est noté $P_{A,C}$. Pour cela, nous appliquons ce qui suit :

1 - Remplacer toutes les actions $x \in X$ et $y \in Y$ par l'action non observable τ,

2 - Déterminer l'automate de traces correspondant,

3 - Si un état dans l'automate de traces contient l'état silencieux de $I_{A,C}$, nous le déclarons comme un état silencieux,

4 - Merger tout les états silencieux en un seul état silencieux.

Proposition 5.2 : Une trace α mène dans $P_{A,C}$ de l'état initial à l'état silencieux si et seulement si il existe une trace α_1 de $I_{A,C}$ telle que $\alpha = \mathrm{Pr}_{X' \cup U \cup Y' \cup Z}(\alpha_1)$, et la trace α_1 mène dans $I_{A,C}$ de l'état initial à l'état silencieux.

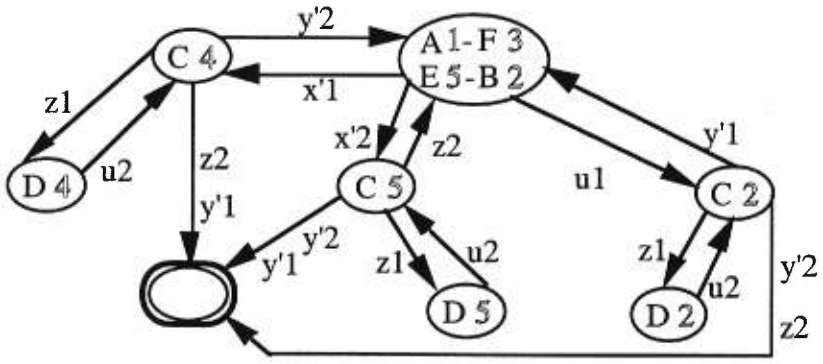


Figure 5.8 : Le système de transition étiquetées $P_{A,C}$

En faite, les traces qui mènent dans $P_{A,C}$ de l'état initial à l'état silencieux représentent toutes les traces interdites de la composante à concevoir.

Étape 4. À partir du système de transition étiquetées $P_{A,C}$, nous construisons une machine de Mealy $[[A,C]]=(S, X' \cup U, Y' \cup Z \cup \{fail\}, h, s_0)$ avec $fail \notin Y' \cup Z$. Cette construction est réalisée de la façon suivante :

1 - L'ensemble des états S de $[[A,C]]$ contient tous les état de $P_{A,C}$ où une transition sortante étiquetées par une action dans $X' \cup U$ est présente, un nouvel état *FAIL* représentant l'état silencieux de $P_{A,C}$ et un état spécial *TRAP*.

2 - Pour toute paire de transitions $s_i \rightarrow v \rightarrow s_j$ et $s_j \rightarrow w \rightarrow s_k$ dans le système de transition étiquetées $P_{A,C}$ où $v \in X' \cup U$ et $w \in Y' \cup Z$, nous associons la transition $s_i \rightarrow v/w \rightarrow s_k$ à la machine de Mealy $[[A,C]]$, c'est à dire, $(s_k, w) \in h(s_i, v)$. Pour tout élément v de $X' \cup U$ nous associons à la machine de Mealy $[[A,C]]$ la transition

FAIL→*v*/*fail*→*FAIL*, c'est à dire, $\forall v \in (X' \cup U)$ $h(FAIL, v)=\{(FAIL, fail)\}$. De plus, pour tout état s dans S et tout v dans $(X' \cup U)$ si $h(s, v)$ n'est pas déja définie alors on pose $h(s, v)=\{(TRAP, w) | w \in (Y' \cup Z)\}$.

3 - s_0 est l'état initial de $P_{A,C}$.

Toute transition non définie est considérée comme une transition "don't care", et nous utilisons l'état *TRAP* de manière similaire à [Unger 69] afin de spécifier formellement de telles situations. En réalité, ces transitions ne seront pas exécutées dans la composition avec le contexte donné. La machine de Mealy [[*A*,*C*]] obtenue à la fin de cette étape est illustrée dans Figure 5.9. Pour alléger la figure, les transitions menant à l'état *TRAP* ne sont pas illustrées.

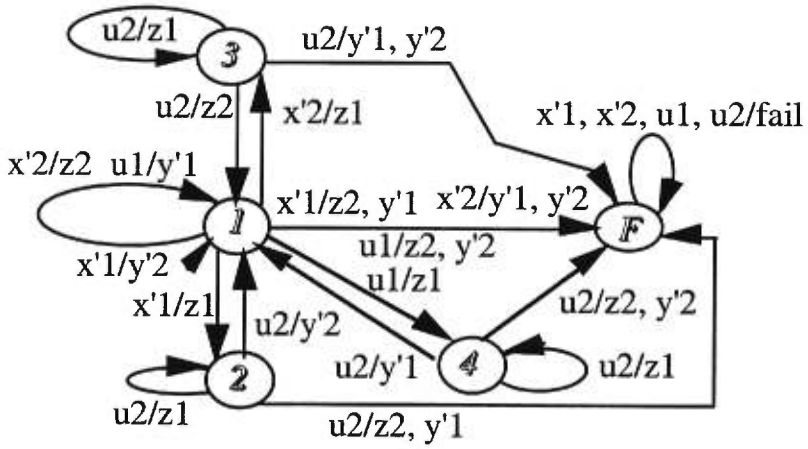


Figure 5.9 : La machine de Mealy [[*A*,*C*]]

Proposition 5.3 : Étant données les séquences $\beta \in (X' \cup U)^*$ et $\delta \in (Y' \cup Z)^*$, la machine de Mealy [[*A*,*C*]] a la séquence d'entrées et sorties β/δ si et seulement si tout préfixe propre de la trace $\beta \bowtie \delta$ qui est une trace du système de transition étiquetées $P_{A,C}$ ne mène pas dans $P_{A,C}$ de l'état initial à l'état silencieux.

Théorème 5.1 : Étant données deux machines de Mealy déterministes complètement spécifiées $A = (S, X \cup X', Y \cup Y', \delta, \lambda, s_0)$ et $C = (S_1, X \cup Z, Y \cup U, \delta_1, \lambda_1, c_0)$. Pour une machine de Mealy déterministe complètement spécifiée $B= (S_2, X \cup Z, Y \cup U, \delta_2, \lambda_2, b_0)$ telle que $C \Diamond B$ existe, la machine de Mealy $C \Diamond B$ est équivalente à A si et seulement si B est une réduction de [[*A*,*C*]].

Preuve du Théorème 5.1

On pose $C \Diamond B = (P, X \cup X', Y \cup Y', \Delta, \Lambda, p_0)$.

Première partie. ($\Leftarrow$)

Si les machines de Mealy $C \Diamond B$ et A ne sont pas équivalentes alors il existe une séquence d'entrées $\alpha \in (X \cup X')^*$ telle que $\lambda(s_0, \alpha) \neq \Lambda(p_0, \alpha)$, c'est à dire, $\lambda(s_0, \alpha) = y_1 ... y_{j-1} y_j$ et $\Lambda(p_0, \alpha) = y_1 ... y_{j-1} \tilde{y}_j$ avec $\tilde{y}_j \neq y_j$. Soit $\beta_1 \tilde{y}_j$ la trace exécutée par le système de transition étiquetées $I_C \| I_B \| I_E$ lorsque α est appliquée à l'état initial de $C \Diamond B$. D'après Proposition 5.1, la trace $\beta_1 \tilde{y}_j$ mène le système de transition étiquetées $I_{A,C}$ de l'état initial à l'état silencieux, c'est à dire, la trace $\mathrm{Pr}_{X' \cup U \cup Y' \cup Z}(\beta_1 \tilde{y}_j)$ mène le système de transition étiquetées $P_{A,C}$ de l'état initial à l'état silencieux (Proposition 5.2), et d'après la construction de la machine de Mealy $[[A,C]]$ (Étape 5), la séquence d'entrées et sorties $\mathrm{Pr}_{X' \cup U}(\beta_1)/\mathrm{Pr}_{Y' \cup Z}(\beta_1 \tilde{y}_j)$ mène de l'état initial à l'état *FAIL* dans $[[A,C]]$.

Ceci entraîne que la machine de Mealy B ne peut être une réduction de $[[A,C]]$ par rapport à aucun prolongement de la séquence d'entrées $\mathrm{Pr}_{X' \cup U}(\beta_1)$ puisque la machine de Mealy $[[A,C]]$ dans l'état *FAIL* produit la sortie *fail* $\notin Y' \cup Z$ pour toute entrée $v \in X' \cup U$.

Deuxième partie. ($\Rightarrow$)

Si la machine de Mealy B n'est pas une réduction de $[[A,C]]$, il existe une séquence d'entrées et sorties v/μ de B qui n'est pas une séquence d'entrées et sorties de $[[A,C]]$. Donc, il existe un préfixe propre β/γ de v/μ tel que la trace $\beta \bowtie \gamma$ mène dans le système de transition étiquetées $P_{A,C}$ de l'état initial à l'état silencieux (Proposition 5.3), c'est à dire, il existe une trace ξ de $I_{A,C}$ telle que $\beta \bowtie \gamma = \mathrm{Pr}_{X' \cup U \cup Y' \cup Z}(\xi)$, et la trace ξ mène dans le système de transition étiquetées $I_{A,C}$ de l'état initial à l'état silencieux (Proposition 5.2). Soit $\xi = \xi_1 y$, alors ξ est une trace du système de transition étiquetées $I_C \| I_B \| I_E$ et $\mathrm{Pr}_{Y \cup Y'}(\xi_1) y'$ est la séquence de sortie de la machine de Mealy A produite en réponse à $\mathrm{Pr}_{X \cup X'}(\xi)$, telle que $y \neq y'$ (Proposition 5.1).

Ceci entraîne que les machines de Mealy $C \Diamond B$ et A ne sont pas équivalente par rapport à la séquence d'entrées $\mathrm{Pr}_{X \cup X'}(\xi)$. □

5.4 Élimination de la sortie *fail*

Nous nous intéressons au problème de la recherche de l'ensemble des solutions potentielles de l'équation $C \Diamond X \cong A$ qui peut être décrit comme l'ensemble des D-réductions d'une machine de Mealy ayant pour ensemble d'entrées $X' \cup U$ et pour ensemble de sorties $Y' \cup Z$, tandis que la machine de Mealy $[[A,C]]$ a pour ensemble de sorties $Y' \cup Z \cup \{fail\}$. Notre prochaine étape consiste à éliminer de l'ensemble des sorties de $[[A, C]]$ la sortie superflue *fail* tout en préservant l'ensemble des réductions déterministes ayant pour ensemble d'entrées $X' \cup U$ et pour ensemble de sorties $Y' \cup Z$ de la machine de Mealy $[[A, C]]$.

Le problème à résoudre est un cas particulier du problème suivant. Étant donnée une machine de Mealy $K=(S, V, W', h, s_0)$ et un sous ensemble W de l'ensemble des sorties W', nous désirons construire une sous-machine D de K ayant pour ensemble de sorties W telle que les ensembles des D-réductions ayant pour ensemble d'entrées V et pour ensemble de sorties W des deux machines de Mealy K et D coïncident.

Soit B une machine de Mealy qui est une D-réduction de K. Pour tout état t de B il existe un état s de K tel que t est une réduction de s. Pour cette raison, si l'on élimine de K chaque état s pour lequel il n'existe aucun état b d'une machine de Mealy déterministe ayant pour ensemble de sortie W tel que $b \leq s$, et toutes les transitions étiquetées par des actions non présentes dans W, la sous-machine résultante préservera toutes les réductions de K ayant pour ensemble d'entrées V et pour ensemble de sorties W. La définition 5.1 permet de caractériser les états de K pour lesquels il n'existe pas d'état b d'une machine de Mealy déterministe ayant pour ensemble de sortie W tel que $b \leq s$.

Définition 5.1 : Étant donnée une machine de Mealy $K=(S, V, W', h, s_0)$ et étant donné un ensemble $W \subset W'$, un état s de K est dit $W(1)$-*redondant* si il existe une entrée $v \in V$ telle que $h^2(s, v) \cap W = \varnothing$. Un état s de K est dit $W(k+1)$-redondant si il est $W(k)$-redondant, ou si il existe une entrée $v \in V$ telle que tous les états dans $h^1(s, v)$ sont $W(k)$-redondant. Un état s est dit W-redondant si il existe un entier k tel que s est $W(k)$-redondant.

Puisque l'ensemble S contenant les états de K est fini, il existe un entier $k \leq |S|$ tel que les ensembles contenant les états $W(k)$-redondant et $W(k+1)$-redondant coïncident. Nous notons $\hat{S}$ l'ensemble des états W-redondant de K.

Proposition 5.4 : Étant donnée une machine de Mealy $K= (S, V, W', h, s_0)$, étant donné un ensemble $W \subset W'$, et un état s de K, si il existe une machine de Mealy déterministe $B = (Q, V, W, \delta, \lambda, q_0)$ et un état q de B tel que $q \leq s$ alors s n'est pas un état W-redondant de K.

Preuve de Proposition 5.4

1. Si il existe un état q d'une machine de Mealy B tel que $q \leq s$, alors l'état s n'est pas $W(1)$-redondant.
2. Hypothèse d'induction : supposons que si il existe un état de B qui est une réduction de l'état s de A alors l'état s n'est pas $W(k\text{-}1)$-redondant pour $k \geq 2$.
3. Supposons maintenant que l'état s est $W(k)$-redondant et que il existe un état q d'une machine de Mealy déterministe B tel que $q \leq s$. Alors il existe une entrée $v \in V$ telle que pour toute sortie $w \in W$ l'état $h_w^1(s,v)$ (si il existe) est $W(j)$-redondant, avec $j<k$. Puisque q est une

réduction de s alors l'état $q'=\delta(q, v)$ de B doit être une réduction de l'état $h^1_w(s,v)$, où $w=\lambda(q,v)\in W$. Ceci est en contradiction avec l'hypothèse d'induction.
Donc, si q est une réduction de l'état s alors s n'est pas $W(k)$-redondant pour tout entier k, c'est à dire, $s\notin \hat{S}$. □

Proposition 5.5 : Étant donnée une machine de Mealy $K = (S, V, W', h, s_0)$ et étant donné un ensemble $W\subset W'$, soit $\hat{S}$ l'ensemble de tous les états W-redondant de K. Si $s_0\in \hat{S}$ alors l'ensemble des réduction de K ayant pour ensemble d'entrées V et pour ensemble de sorties W est vide. Si $s_0\notin \hat{S}$, la sous-machine $D =(S\setminus\hat{S}, V, W, \hat{h}, s_0)$ de K, telle que $\hat{h}(s,v) = h(s,v)\setminus[\{(s',w')|\ w'\notin W\ \}\cup\{(s',w)|\ s'\in \hat{S}\ \}]$, pour tout $(s,v)\in(S\setminus\hat{S})\times V$ a le même ensemble de D-réductions ayant pour ensemble d'entrées V et pour ensemble de sorties W que la machine de Mealy K.

Preuve de Proposition 5.5
Si $s_0\in \hat{S}$ alors il n'existe pas d'état b d'une quelconque machine de Mealy déterministe ayant pour ensemble d'entrées V et pour ensemble de sorties W tel que $b\leq s_0$ (Proposition 5.4), et donc, l'ensemble des réduction de K ayant pour ensemble d'entrées V et pour ensemble de sorties W est vide. Si $s_0\notin \hat{S}$ alors, d'après la définition de l'ensemble $\hat{S}$, l'ensemble $\hat{h}(s, v)$ n'est pas vide pour chaque $(s,v)\in(S\setminus\hat{S})\times V$. De plus, par construction de $\hat{h}$, $\hat{h}(s,v)\subseteq(S\setminus\hat{S})\times W$. Donc, D est une sous-machine de K.
Soient $B = (Q, V, W, \delta, \lambda, q_0)$ une D-réduction de K et $w_1\ldots w_k$ la séquence de sorties de B correspondante à la séquence d'entrées $v_1\ldots v_k$ lorsque appliquée à partir de l'état initial. L'état $\delta(q_0, v_1\ldots v_j)$ est une réduction de l'état $h^1_{w1\ldots wj}(s_0, v_1\ldots v_j)$ pour tout $j=1,\ldots,k$, c'est à dire, l'état $h^1_{w1\ldots wj}(s_0, v_1\ldots v_j)\in S\setminus\hat{S}$ (Proposition 5.4).
Donc, $(\delta(q_0, v_1\ldots v_j), w_j)\in\hat{h}(s_0, v_1\ldots v_j)$, ce qui entraîne que B est une réduction de D. □

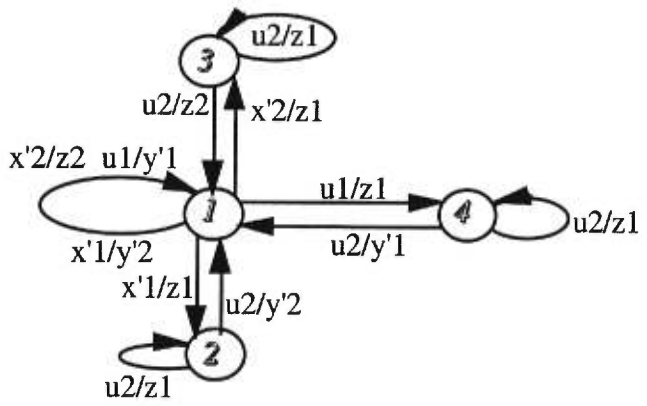


Figure 5.10 : La machine de Mealy $[[A, C]]_f$

À partir de Proposition 5.5, on développe l'algorithme suivant qui permet de dériver à partir de la machine de Mealy $[[A, C]]$ ayant pour ensemble de sorties $Y' \cup Z \cup \{fail\}$ une machine de Mealy ayant pour ensemble de sorties $Y' \cup Z$, on la note $[[A, C]]_f$, qui a le même ensemble de D-réductions ayant pour ensemble d'entrées $X' \cup U$ et pour ensemble de sorties $Y' \cup Z$ que la machine de Mealy $[[A, C]]$. Pour notre exemple, la machine de Mealy $[[A, C]]_f$, est illustrée dans Figure 5.10. Pour alléger la figure, les transitions menant à l'état *TRAP* ne sont pas illustrées.

Algorithme

Entrée : La machine de Mealy $[[A,C]] = (S, X' \cup U, Y' \cup Z \cup \{fail\}, h, s_0)$.

Sortie : Une sous-machine $[[A, C]]_f$ de $[[A, C]]$ ayant pour ensemble d'entrées $X' \cup U$ et pour ensemble de sorties $Y' \cup Z$ qui a le même ensemble de D-réductions ayant pour ensemble d'entrées $X' \cup U$ et pour ensemble de sorties $Y' \cup Z$ que la machine de Mealy $[[A, C]]$ si l'ensemble des D-réductions de $[[A, C]]$ n'est pas vide.

Étape 1. Construire l'ensemble $\hat{S}$ des états $Y' \cup Z$-redondant de $[[A, C]]$. Si $s_0 \in \hat{S}$ alors la machine de Mealy $[[A, C]]$ n'a pas de D-réductions ayant pour ensemble de sorties $Y' \cup Z$, et la procédure se termine; sinon on passe à Étape 2.

Étape 2. Pour tout $(s, v) \in (S \setminus \hat{S}) \times X' \cup U$, $\hat{h}(s, v)$ est obtenu à partir de $h(s, v)$ en éliminant chaque paire (s', w) telle que $s' \in \hat{S}$ ou $w \notin Y' \cup Z$. La sous-machine recherchée est définie de la façon suivante : $[[A, C]]_f = (S \setminus \hat{S}, X' \cup U, Y' \cup Z, \hat{h}, s_0)$.

5.5 L'ensemble des solutions de l'équation

Le théorème suivant permet de caractériser les solution de l'équation $C \Diamond X \cong A$ à partir de la machine de Mealy $[[A, C]]_f$.

Théorème 5.2. Étant données deux machines de Mealy déterministes complètement spécifiées $A = (S, X \cup X', Y \cup Y', \delta, \lambda, s_0)$ et $C = (S_1, X \cup Z, Y \cup U, \delta_1, \lambda_1, c_0)$. Si la machine de Mealy $[[A, C]]_f$ existe alors l'ensemble des solutions de l'équation $C \Diamond X \cong A$ coïncide avec l'ensemble des D-réductions B de $[[A, C]]_f$ pour lesquelles $C \Diamond B$ existe.

Preuve du Théorème 5.2

La preuve de ce théorème découle directement du Théorème 5.1 et de la Proposition 5.5.

Dans le cas où la machine de Mealy $[[A, C]]_f$ n'existe pas, il n'y a aucune solution pour l'équation $C \Diamond X \cong A$. Quand la machine de Mealy $[[A, C]]_f$ existe, deux cas sont

possibles. Dans le premier cas, le système de transitions étiquetées $I_C \| I_{[[A, C]]_f} \| I_E$ ne contient pas de cycle étiquetés seulement par des actions internes (livelocks). Dans ce cas, toute D-réduction de $[[A, C]]_f$ est une solution de l'équation $C \Diamond X \cong A$, et $[[A, C]]_f$ est la solution générique de l'équation. Dans le deuxième cas, le système de transitions étiquetées $I_C \| I_{[[A, C]]_f} \| I_E$ contient des cycles étiquetés seulement par des actions internes. Il y a alors deux possibilités :

1- toute D-réduction F de $[[A, C]]_f$ n'est pas une solution de l'équation car $C \Diamond F$ n'existe pas à cause de la présence de cycle étiquetés seulement par des actions internes dans $I_C \| I_F \| I_E$ (voir Exemple 1, Figure 5.11 et Figure 5.12).

2- l'ensemble des solutions de l'équation $C \Diamond X \cong A$ est un sous ensemble de l'ensemble des D-réductions de $[[A, C]]_f$ (voir Exemple 2, Figure 5.13 et Figure 5.14).

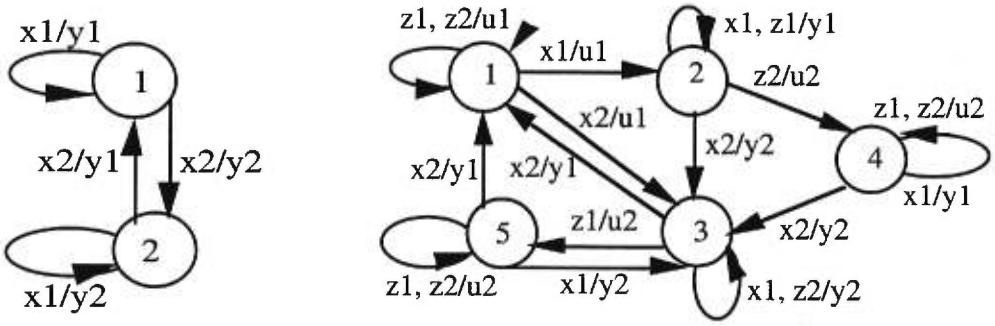


Figure 5.11 : Les machines de Mealy A, C

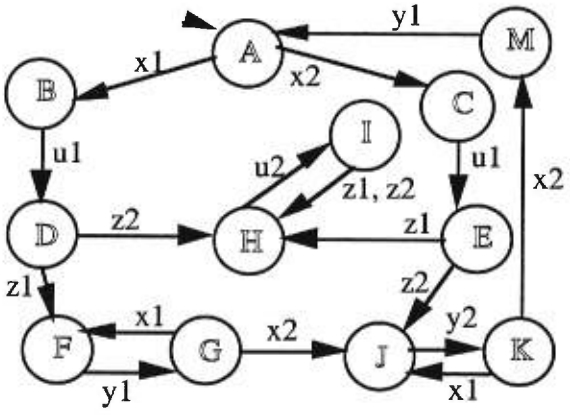


Figure 5.12 : Le système de transitions étiquetées $I_C \| I_{Ch} \| I_E$

Exemple 1

La machine de Mealy $[[A, C]]_f$ obtenue dans le cas de l'exemple 1 est la machine chaotique. La machine $C \Diamond F$ n'existe pour aucune D-réduction F de $[[A, C]]_f$ car après

n'importe quelle sortie produite par F en réponse à l'entrée u_1 dans l'état initial de F, le système composé entre dans un cycle étiquetés seulement par des actions internes après la réception dans l'état initial de soit l'entrée x_1 soit l'entrée x_2. Cette exemple illustre donc bien le cas où la machine $[[A, C]]_f$ existe alors qu'aucune de ses D-réductions n'est solution de l'équation.

Exemple 2

Dans cette exemple, nous considérons la même machine A décrite dans Exemple 1 par contre le contexte C est celui illustré dans Figure 5.13.

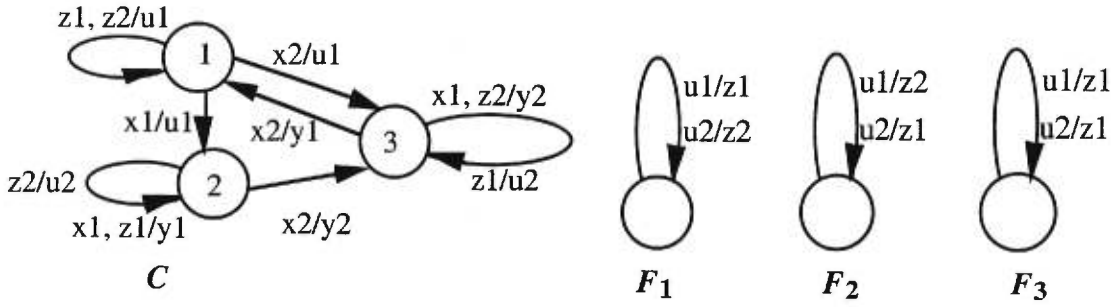


Figure 5.13 : Les machines de Mealy C, F_1, F_2 et F_3

La machine de Mealy $[[A, C]]_f$ obtenue dans le cas de l'exemple 2 est aussi la machine chaotique. Par contre ses D-réductions F_1 et F_2 sont des solutions de l'équation, alors que sa D-réduction F_3 n'est pas une solution de l'équation. De plus dans cette exemple, la solution générique n'existe pas car pour toute séquence d'entrées et sorties de $[[A, C]]_f$ nous pouvons trouver une D-réduction F de $[[A, C]]_f$ qui est une solution de l'équation et qui contient la séquence d'entrées et sorties. Considérons la machine de Mealy B_n illustrée dans Figure 5.14 :

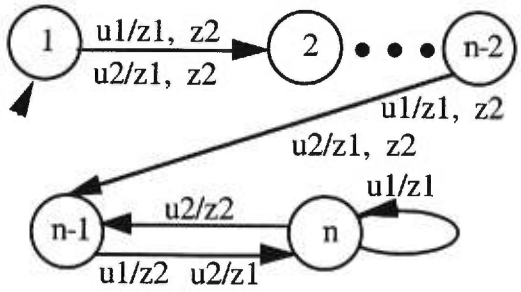


Figure 5.14 : La machine de Mealy B_n

La machine de Mealy B_n est une réduction de $[[A, C]]_f$ et le système de transitions étiquetées $I_C \| I_{B_n} \| I_E$ ne contient pas de cycles étiquetés seulement par des actions internes. Donc, toute D-réduction de B_n, $n>1$, est une solution de l'équation $C \Diamond X \cong A$. Considérons maintenant n'importe quelle séquence d'entrées et sorties sur l'ensemble d'entrées U et l'ensemble de sorties Z de longueur m. C'est une séquence d'entrées et sorties de la machine de Mealy B_{m+2} et si l'on choisit une sous-machine déterministe F de B_{m+2} contenant cette séquence d'entrées et sorties alors F est une solution de l'équation $C \Diamond X \cong A$. Donc si une solution générique G de l'équation existe, nous aurons $G \leq [[A, C]]_f$ et $[[A, C]]_f \not\leq G$, mais si l'on élimine n'importe quelle trace de $[[A, C]]_f$ nous perdons certaines solutions de l'équation. Ceci entraîne qu'il n'existe pas de solution générique dans ce cas.

5.6 Conclusion

Nous avons présenté dans ce chapitre une approche pour la résolution du problème de construction de sous-modules dans le cas où les spécifications du système désiré et du contexte sont données sous la forme de Machines de Mealy déterministes complètement spécifiées. La relation de conformité utilisée est la relation trace équivalence. L'ensemble des solutions de l'équation (lorsqu'il n'est pas vide) peut être décrit par l'ensemble des D-réductions pour lesquelles la composition avec le contexte existe d'une machine de Mealy non déterministe. La nécessité de supposer que la composition existe indique que la relation de réduction est insuffisante pour garantir qu'une machine est une solution.

Une généralisation directe de ce travail au cas des machines de Mealy non déterministes et complètement spécifiées se trouve dans [Petrenko 98]. Il présente de plus une approche pour traiter le problème des cycles étiquetés seulement par des actions internes. Ce problème peut être décrit de la façon suivante : après la réception d'une entrée externe par le système, le contexte et la composante peuvent avoir un nombre non borné d'interactions internes successives. L'approche est alors basée sur la limitation du nombre d'interactions internes successives permises entre le contexte et la composante à concevoir. Le cas des machines de Mealy partiellement spécifiées restait ouvert dans ce travail. Dans la section 4 du chapitre 7, nous présenterons une méthode pour la résolution du problème de construction de sous-modules dans ce dernier cas.

CHAPITRE 6

GÉNÉRATION D'UNE IMPLANTATION DÉTERMINISTE MINIMALE À PARTIR D'UNE SPÉCIFICATION NON DÉTERMINISTE

Dans le chapitre précédant, la solution générique, lorsqu'elle existe, est décrite sous la forme d'une machine de Mealy non déterministe. L'implanteur peut chercher à réduire son non déterminisme afin d'obtenir une implantation déterministe dont le nombre d'états est minimal. Dans ce chapitre, nous proposons une méthode pour la détermination d'une implantation minimale dont les traces sont incluses dans celles d'une spécification non déterministe. Dans la Section 6.1, on s'intéresse aux conditions pour merger plusieurs états en un seul, ce qui mène à la définition de classe machine compatible. Nous montrons par la suite que toute réduction déterministe correspond à un ensemble de classes machine compatibles, puis on s'intéresse à la minimalité de ces ensembles. La Section 6.2 illustre des méthodes pour la construction d'une réduction déterministe minimale.

6.1 Caractérisation de la réduction déterministe minimale

6.1.1 Description du problème

Dans ce travail, nous nous sommes inspirés des travaux sur la réduction des machines déterministes partiellement spécifiées [Kohavi 78][Grasselli 65]. La méthode utilisée pour la réduction est composée des trois étapes principales suivantes : la définition et l'identification des états compatibles, la construction d'ensembles fermés de classes compatibles et le choix d'un ensemble fermé contenant le nombre minimal de classes. Au cours de ce travail, nous adaptons et généralisons chacune des étapes à la classe des machines non déterministes.

Étant donnée une machine de Mealy observable complètement spécifiée non déterministe $A=(S, X, Y, h, s_0)$, on s'intéresse à la recherche d'une machine de Mealy déterministe $B=(T, X, Y, \delta, \lambda, t_0)$ qui soit une réduction de A , et qui a un nombre d'états inférieur ou égale au nombre d'états de toute D-réduction de A. La machine B est dite une D-réduction de A. On ne considère que les machines observables, car toute machine a une forme observable.

Évidement, toutes les D-sous-machines (voir 2.3 pour la notion de D-sous-machines) de A sont des réductions de A, Mais il peut exister des D-réductions de A qui ne sont pas des D-sous-machines de A. L'exemple suivante extrait de [Petrenko 93] illustre ce cas.

Exemple :

Dans cette exemple, nous considérons les machines de Mealy A et B (Figure 6.1). Il est facile de vérifier que B est une D-réduction de A. Par contre B n'est équivalente à aucune sous-machine de A, car après la réception de l'entrée a la machine B se retrouve toujours dans son état 1 alors que la machine A ne change pas son état.

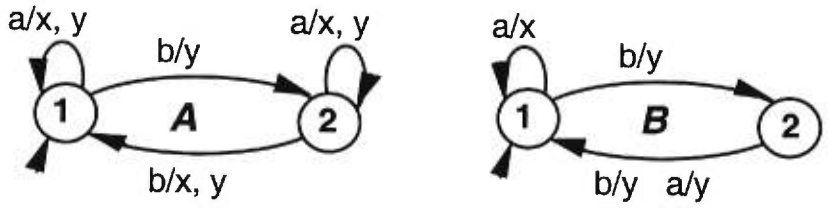


Figure 6.1 : Les machines de Mealy A et B

Pour cette raison, le problème ne peut être résolu directement, par exemple, par l'élimination de certaines transitions, comme dans [Qin 91]. Mais en même temps, comme nous le montrerons plus loin dans ce chapitre, il est possible de construire une machine dont l'ensemble des D-sous-machines contient toutes les D-réductions de A.

Si B est une D-réduction de A alors tout état de B doit être une réduction d'un état approprié de A. Puisqu'un état de B peut être une réduction de plusieurs états de A, nous nous intéressons d'abord aux conditions sous lesquelles ceci est possible.

De façon intuitive, pour qu'un état de B soit une réduction d'un sous ensemble d'états de A, il est nécessaire que tous les états de ce sous ensemble produisent au moins une séquence de sorties commune pour chaque séquence d'entrées possible. Cette condition est

utilisée dans la définition d'une classe compatible donnée dans [Damiani 94]. Mais la condition est non suffisante comme nous l'illustrerons dans l'exemple dans Figure 6.2.

Définition 6.1 [Damiani 94] **:** Soit C un sous ensemble d'états de A. C est une *classe compatible* si pour toute séquence d'entrées $\alpha \in X^*$ $\bigcap_{s \in C} h^2(s, \alpha) \neq \emptyset$.

L'idée derrière cette définition est la détermination d'un ensemble d'états d'une machine de Mealy observable complètement spécifiée non déterministe A pouvant être regroupés en un seul état d'une D-réduction de A. Or cette généralisation directe de la notion de classe compatible pour les machines déterministes n'est pas adéquate pour le cas non-déterministe. D'ailleurs comme le prouve l'exemple suivant, un ensemble d'états qui satisfait la définition 6.1 ne correspond pas nécessairement à un état d'une machine déterministe.

Un exemple d'une machine de Mealy non déterministe A est donné dans la Figure 6.2. Le sous ensemble d'états C_1={2, 3} de A est une classe compatible selon la Définition 6.1. Le théorème 4.2 de [Damiani 94] stipule que si un ensemble d'états C est une classe compatible alors pour toute entrée x il existe au moins une sortie y telle que C'={c | s-x/y->c et $s \in C$} est une classe compatible. Or, pour la classe compatible C_1 et l'entrée a, aucun des sous ensembles C_2={c | s-a/0->c et $s \in C_1$}={4, 6} et C_3={c | s-a/1->c et $s \in C_1$}={4, 5} ne forme une classe compatible. Les états 2 and 3 ne peuvent donc pas correspondre à un seul état d'une machine de Mealy déterministe. Cette exemple illustre un cas de non applicabilité du travail de [Damiani 94].

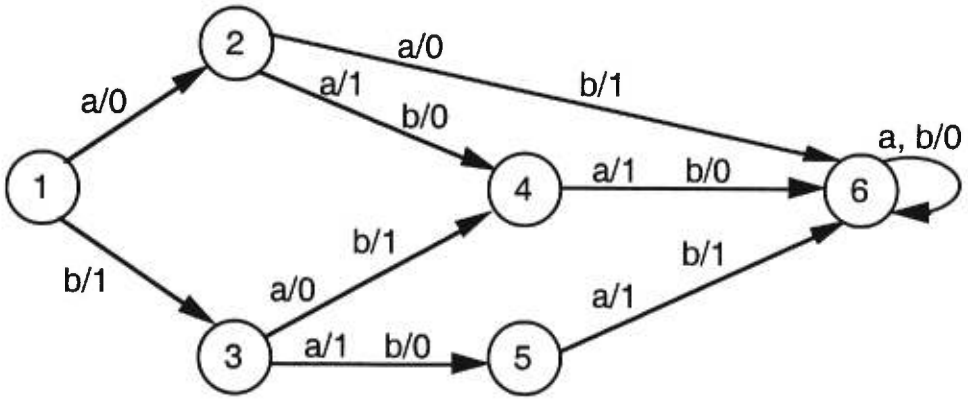


Figure 6.2 : La machine de Mealy A

6.1.2 Classe M-compatible

Pour garantir que plusieurs états d'une machine non déterministe puissent être

implantés par un seul état d'une machine déterministe, nous introduisons la notion de classe machine compatible. Pour deux séquences d'entrées/sorties β/ψ et α/γ, on note $\beta/\psi < \alpha/\gamma$, si β est préfixe propre de α et ψ est préfixe propre de γ.

Définition 6.2 : Soient A une machine de Mealy observable complètement spécifiée non déterministe et C un sous ensemble d'états de A. C est dit une classe *machine compatible* (classe *M-compatible*) si, pour tout $\alpha \in X^*$, il existe une séquence non vide $\gamma \in \bigcap_{s \in C} h^2(s, \alpha)$ telle que l'ensemble $\{p \mid s - \beta/\psi \rightarrow p \ \& \ s \in C\}$ est une classe M-compatible pour toute séquence d'entrées/sorties $\beta/\psi < \alpha/\gamma$. Deux états ou plus sont dit des *états M-compatible* si ils forment une partie d'une même classe M-compatible C.

La notion introduite de classe M-compatible est une généralisation de celle pour des machines partiellement spécifiées, voir par exemple, [Kohavi 78]. Elle diffère de celle du cas déterministe par le fait que trois états ou plus qui sont deux à deux M-compatibles ne forment pas nécessairement une classe M-compatible de la machine non déterministe.

Lemme 6.1 : Soient $A=(S, X, Y, h, s_0)$ une machine de Mealy observable complètement spécifiée non déterministe et $B = (T, X, Y, \delta, \lambda, t_0)$ une machine de Mealy déterministe. B est une D-réduction de A, $B \leq A$, si et seulement si s_0 et t_0 sont des états M-compatibles

Étant donnée une machine de Mealy observable complètement spécifiée non déterministe A, un sous ensemble d'états C et une séquence d'entrées/sorties α/γ, telle que $\gamma \in \bigcap_{s \in C} h^2(s, \alpha)$, on note NEXT($C$, α/γ), l'ensemble des états de A accessibles par α/γ à partir d'un état dans C, c'est à dire

$$\text{NEXT}(C, \alpha/\gamma) = \{c \mid s - \alpha/\gamma \rightarrow c \ \& \ s \in C\} = \bigcup_{s \in C} h^1_\gamma(s, \alpha).$$

Pour pouvoir fusionner un ensemble d'états C en un seul état, il est nécessaire que pour chaque entrée, il existe au moins une sortie telle que les états accessibles par cette entrée/sortie à partir des états de C puissent aussi être mergés en un seul état.

Proposition 6.1 : Soit C un sous ensemble d'états d'une machine de Mealy observable complètement spécifiée non déterministe A. C est une classe M-compatible si et seulement si pour tout $x \in X$ il existe $y \in Y$ tel que NEXT(C, x/y) est une classe M-compatible.

Preuve de Proposition 6.1

C est une classe M-compatible

$\Leftrightarrow$ pour tout $x \in X$ il existe $y \in Y$ tel que

$$\forall x\alpha \in X^* \; \exists\, y\gamma \in \bigcap_{s \in C} h^2(s, x\alpha) \; \forall \beta \in X^* \; \exists \sigma \in Y^* \; (y\gamma\sigma \in \bigcap_{s \in C} h^2(s, x\alpha\beta))$$

$\Leftrightarrow$ pour tout $x \in X$ il existe $y \in Y$ tel que

$$\forall \alpha \in X^* \; \exists \gamma \in \bigcap_{s' \in \mathrm{NEXT}(C, x/y)} h^2(s', \alpha) \; \forall \beta \in X^* \; \exists \sigma \in Y^* \; (\gamma\sigma \in \bigcap_{s' \in \mathrm{NEXT}(C, x/y)} h^2(s', \alpha\beta))$$

$\Leftrightarrow$ pour tout $x \in X$ il existe $y \in Y$ tel que NEXT(C, x/y) est une classe M-compatible. □

6.1.3 Système M-compatible

À ce niveau les états qui peuvent être regroupés ensemble sont identifiés. Mais plusieurs façons de les regrouper sont possibles. Un regroupement peut être représenté par un ensemble de classes M-compatibles. Ceci nous mène à la définition suivante.

Définition 6.3 : Soit $CS=\{C_0, C_1, ..., C_{k-1}\}$ un ensemble de classes M-compatibles d'une machine de Mealy observable complètement spécifiée non déterministe A

(1) CS est appelé un *système M-compatible* ;

(2) CS est dit fermé s'il existe $l \in \{0,..., k-1\}$ tel que $s_0 \in C_l$, et pour tout $C_i \in CS$ et toute entrée x, il existe une sortie $y \in \bigcap_{s \in C_i} h^2(s, x)$ et une classe $C_j \in CS$ telles que NEXT(C_i, x/y)$\subseteq C_j$;

(3) CS est dit un *système M-compatible fermé minimal* s'il ne contient pas strictement un système M-compatible fermé.

Ces concepts sont une généralisation naturelle de concepts similaires de la théorie des machines déterministes partiellement spécifiées [Kohavi 78]. Nous donnons dans la définition suivante une correspondance entre les systèmes M-compatibles et des machines déterministes.

Définition 6.4 : Soit $CS=\{C_0, C_1, ..., C_{k-1}\}$ un système M-compatible fermé d'une machine de Mealy observable complètement spécifiée non déterministe $A=(S, X, Y, h, s_0)$ tel que $C_0 \ni s_0$. Une machine de Mealy déterministe $B=(T, X, Y, \delta, \lambda, t_0)$ est dite *correspondre* à CS s'il existe une bijection μ: $T \rightarrow CS$, telle que

- $\mu(t_0)=C_0$
- $\lambda(t, x) \in \bigcap_{s \in \mu(t)} h^2(s, x)$ et NEXT($\mu(t)$, $x/\lambda(t, x)$)$\subseteq \mu(\delta(t, x))$ pour tout $t \in T$ et $x \in X$.

Il est important maintenant de montrer que les machines déterministes obtenues sont

bien des réductions de notre machine de départ.

Proposition 6.2 : Soient une machine de Mealy observable complètement spécifiée non déterministe A et un système M-compatible fermé CS de A. Toute machine de Mealy déterministe B correspondant à CS est une D-réduction de A.

Preuve de Proposition 6.2

Soit $A=(S, X, Y, h, s_0)$ une machine de Mealy observable complètement spécifiée non déterministe et $B=(T, X, Y, \delta, \lambda, t_0)$ une machine de Mealy déterministe correspondant à un système M-compatible fermé $CS=\{C_0, C_1, ..., C_{k-1}\}$ avec $\mu(t_0)=C_0$. Selon le Lemme 6.1, nous devons prouver que s_0 et t_0 sont des états M-compatibles. Nous utilisons une preuve par induction sur la longueur d'une séquence d'entrées.

Base d'induction. Si α est la séquence vide ε alors

$$\lambda(t_0, \varepsilon)=\varepsilon \in \bigcap_{s\in\mu(t_0)} h^2(s, \varepsilon) \text{ et } \mathrm{NEXT}(\mu(t_0), \varepsilon/\varepsilon)=\mu(t_0)\subseteq\mu(\delta(t_0, \varepsilon)).$$

Hypothèse d'induction. Supposons que pour tout $\alpha\in X^*$ de longueur inférieure ou égale à k nous avons $\lambda(t_0, \alpha) =\gamma \in \bigcap_{s\in\mu(t_0)} h^2(s, \alpha)$ et $\mathrm{NEXT}(\mu(t_0), \alpha/\gamma)\subseteq\mu(t)=\mu(\delta(t_0,\alpha))$.

Considérons une entrée $x\in X$ avec $\lambda(t, x)=y$ et $\delta(t, x)=t'$.

Alors $y\in \bigcap_{s\in\mu(t)} h^2(s, \alpha)$ and $\mathrm{NEXT}(\mu(t), x/y)\subseteq \mu(t')$.

D'après la définition d'une machine de Mealy observable non déterministe,

$$\bigcap_{s\in\mu(t_0)} h^2(s, \alpha x)=\{\beta y' \mid \beta\in \bigcap_{s\in\mu(t_0)} h^2(s, \alpha) \;\&\; y'\in \bigcap_{s'\in \mathrm{NEXT}(\mu(t_0),\, \alpha/\beta)} h^2(s', x)\}$$

D'après l'hypothèse d'induction, $\mathrm{NEXT}(\mu(t_0), \alpha/\gamma)\subseteq\mu(t)$ et $\gamma\in \bigcap_{s\in\mu(t_0)} h^2(s, \alpha)$

Donc la séquence $\gamma y\in \bigcap_{s\in\mu(t_0)} h^2(s, \alpha x)$.

De plus, $\mathrm{NEXT}(\mu(t_0), \alpha x /\gamma y)=\mathrm{NEXT}(\mathrm{NEXT}(\mu(t_0), \alpha/\gamma), x/y)\subseteq$

$$\mathrm{NEXT}(\mu(t), x/y)\subseteq\mu(t')=\mu(\delta(t_0, \alpha x)$$

Donc la propriété est vraie pour toute séquence de longueur $(k+1)$.

Nous concluons que B est une D-réduction de A. ❐

Nous décrivons dans ce qui suit, une procédure pour construire une machine de Mealy déterministe $B=(T, X, Y, \delta, \lambda, t_0)$ correspondant à un système M-compatible fermé $CS=\{C_0, C_1, ..., C_{k-1}\}$ tel que $C_0\ni s_0$ d'une machine de Mealy observable complètement spécifiée non déterministe $A=(S, X, Y, h, s_0)$. Soit $T=\{t_0, t_1, ..., t_{k-1}\}$ l'ensemble des états de B, on considère une bijection : μ: $T \rightarrow CS$.

Procedure construction-machine-de-Mealy-déterministe

Entrée : $A=(S, X, Y, h, s_0)$, $CS=\{C_0, C_1, ..., C_{k-1}\}$.

Sortie : $B=(T, X, Y, \delta, \lambda, t_0)$ correspondant à $CS=\{C_0, C_1, ..., C_{k-1}\}$.

- $t_0:=\mu^{-1}(C_0)$;
- CR:=$CS \setminus \{C_0\}$;
- TR:=$T \setminus \{t_0\}$;
- TM:=$\{t_0\}$;
- comp:=1
- Tant que TM≠Ø Faire

pour le premier élément t in TM

pour chaque $x \in X$

on choisit $y \in Y$ et C_j telles que :

$$y \in \bigcap_{s \in \mu(t)} h^2(s, x) \text{ et } \mathrm{NEXT}(\mu(t), x/\lambda(t, x)) \subseteq C_j$$

on pose $\lambda(t, x):=y$

si $C_j \in$ CR alors $\delta(t, x):=t_{comp}$;

$t_{comp}:=\mu^{-1}(C_j)$;

comp:=comp+1;

CR:=CR$\setminus\{C_j\}$;

TM:=TM$\cup\{t_{comp}\}$;

sinon $\delta(t, x):=\mu^{-1}(C_j)$;

TM:=TM$\setminus\{t\}$;

Fin construction-machine-de-Mealy-déterministe

6.1.4 La minimalité

Nous relions dans ce qui suit la notion de minimalité d'un système M-compatible fermé et celle d'une réduction déterministe.

Définition 6.5 : Une machine de Mealy déterministe $B = (T, X, Y, \delta, \lambda, t_0)$ est dite une *D-réduction minimale* d'une machine de Mealy observable complètement spécifiée non déterministe A si il n'existe aucune réduction déterministe de A ayant un nombre d'états inférieur à $|T|$.

Théorème 6.1 : Soit A une machine de Mealy observable complètement spécifiée non déterministe ayant n états. A possède un système M-compatible fermé contenant au plus m classes, avec $m \leq n$, si et seulement si A possède une D-réduction B ayant m états.

Preuve du Théorème 6.1

Première partie. Voir la Procédure "construction-machine de Mealy déterministe" et la Proposition 3.2.

Deuxième partie.

Pour tout état t de B qui est accessible à partir de l'état initial, nous considérons le sous ensemble C_t d'états de A

$$C_t = \{ s \mid \exists \alpha \in X^* \ \delta(t_0, \alpha) = t \ \& \ s_0\text{-}\alpha/\lambda(t_0, \alpha)\text{->}s \ \}.$$

Nous montrons que : $CS = \{ C_t \mid t$ est accessible à partir de l'état initial dans $B \}$ est un système M-compatible fermé.

L'état t_0 est accessible à partir de l'état initial grâce au mot vide ε, donc $C_{t_0} \in CS$.

De plus pour $\alpha = \varepsilon$ nous avons $\delta(t_0, \varepsilon) = t_0$ et $s_0\text{-}\varepsilon/\lambda\ (t_0, \varepsilon)\text{->}s_0$, donc $s_0 \in C_{t_0}$.

Considérons $x \in X$ et $C_t \in CS$, $t \in T$. Posons $\delta\ (t, x) = t'$ et $\lambda\ (t, x) = y$.

Puisque B est une réduction de A, l'état t de B est réduction de tout état $s \in C_t$.

Donc, $\lambda\ (t, x) \in h^2(s, x)$ pour tout $s \in C_t$, ceci entraîne que $\lambda\ (t, x) = y \in \bigcap_{s \in C_t} h^2(s, x)$,

De plus, $\mathrm{NEXT}(C_t, x/y) = \{ s' \mid s\text{-}x/y\text{->}s' \ \& \ s \in C_t \}$,

Donc $\mathrm{NEXT}(C_t, x/y) = \{ s' \mid \exists \alpha \in X^* \ \delta(t_0, \alpha) = t \ \& \ s_0\text{-}\alpha/\lambda(t_0, \alpha)\text{->}s \ \& \ s\text{-}x/y\text{->}s' \}$,

Donc $\mathrm{NEXT}(C_t, x/y) = \{ s' \mid \exists \alpha \in X^* \ \delta(t_0, \alpha x) = t' \ \& \ s_0\text{-}\alpha x/\lambda\ (t_0, \alpha)y\text{->}s' \} = C_{t'}$.

Nous concluons que CS est un système M-compatible fermé dont le nombre d'éléments est au plus égal au nombre d'états dans la machine B. ❐

Corollaire 6.1 : Soit B une machine de Mealy déterministe correspondant à un système M-compatible fermé minimal CS d'une machine de Mealy observable complètement spécifiée non déterministe A, alors toute D-réduction de A a au moins autant d'états que B.

Maintenant, on s'intéresse à construire une machine dont l'ensemble des sous machines contient toutes les réductions déterministe d'une machine de Mealy observable complètement spécifiée non déterministe donnée.

Soit CS un système M-compatible fermé d'une machine de Mealy observable complètement spécifiée non déterministe $A = (S, X, Y, h, s_0)$ tel que $C_0 \ni s_0$. Par définition d'un système M-compatible fermé, il est possible de construire une machine de Mealy complètement spécifiée non déterministe

$$A_{cs} = (CS, X, Y, h_c, C_0)$$

telle que pour tout $C \in CS$ et tout $x \in X$

$$h_c(C, x) = \{ (C', y) \mid y \in \bigcap_{s \in C} h^2(s, x) \ \& \ \mathrm{NEXT}(C, x/y) \subseteq C' \}.$$

On remarque à ce niveau que la machine de Mealy A_{cs} peut être une machine non observable.

Corollaire 6.2 : Soit une machine de Mealy observable complètement spécifiée non déterministe $A = (S, X, Y, h, s_0)$. Une machine de Mealy déterministe $B = (T, X, Y, \delta, \lambda, t_0)$ est une réduction de A si et seulement si B est isomorphe à une D-sous-machine de la machine de Mealy complètement spécifiée non déterministe $A_{cs} = (CS, X, Y, h_0, C_0)$ pour un système M-compatible fermé approprié CS.

Définition 6.6 : Une classe M-compatible C' est *impliquée* par une classe M-compatible C s'il existe une entrée x et une sortie y telle que C'=NEXT(C, x/y). Si pour chaque entrée x on choisit un tel C' alors la collection obtenue après le retrait de toutes les classes non-maximales est appelée un *ensemble impliqué* par C et on le note IS(C).

Définition 6.7 : Si une classe C_j est incluse dans une classe C_i et pour chaque ensemble impliqué IS(C_j) il existe un ensemble impliqué IS(C_i) tel que :

$$\forall C \in \mathrm{IS}(C_i)\ \exists C' \in \mathrm{IS}(C_j)\ (C \subseteq C'),$$

alors la classe C_j est *exclue* par C_i.

Proposition 6.3 : Soit $CS=\{C_0, C_1, ..., C_{k-1}\}$ un système M-compatible fermé tel que $C_0 \ni s_0$. Si C_i est exclue par C_i' alors le système $CS'=(CS \setminus \{C_i\}) \cup \{C_i'\}$ est un système M-compatible fermé.

Preuve de Proposition 6.3

Si i=0 alors $s_0 \in C_i'$ et $C_i' \in CS'$, sinon $s_0 \in C_0$ avec $C_0 \in CS'$.

Pour tout $C_l \in CS$ et toute entrée x, il existe une sortie $y \in \bigcap_{s \in C_l} h^2(s, x)$ et une classe $C_j \in CS$ telle que NEXT(C_l, x/y)$\subseteq C_j$

Donc d'après la Définition 6.7 pour tout $C_l \in CS'$ et toute entrée x, il existe une sortie $y \in \bigcap_{s \in C_l} h^2(s, x)$ et une classe $C_j \in CS'$ telle que NEXT(C_l, x/y)$\subseteq C_j$

En conclusion, CS' est un système M-compatible fermé. ❐

Définition 6.8 : Une classe M-compatible est dite première si elle n'est exclue par aucune autre classe M-compatible.

Proposition 6.4 : Il existe au moins un système M-compatible fermé minimal dont les éléments sont des classes premières.

Preuve de Proposition 6.4

Supposons qu'il existe un système $CS=\{C_1, ..., C_j, ..., C_k\}$ M-compatible fermé minimal et que C_j n'est pas une classe M-compatible première. Alors il existe une classe M-compatible première C_i' qui exclue C_j, et l'ensemble CS' obtenu à partir de CS en remplaçant C_j par C_i' est aussi un système M-compatible fermé minimal. En répétant cette opération pour toute classe non première dans CS, nous obtenons un système M-compatible fermé minimal dont les éléments sont des classes premières. ❐

Soit CS_{pr} l'ensemble de toutes les classes premières d'une machine de Mealy observable complètement spécifiée non déterministe A et $A_{CS_{pr}}$ la machine de Mealy complètement spécifiée non déterministe construite en utilisant CS_{pr}.

Théorème 6.2 : Soit une machine de Mealy observable complètement spécifiée non déterministe A. Si B est une D-sous-machine minimale de $A_{CS_{pr}}$ alors B est une D-réduction minimale de A.

Preuve du Théorème 6.2

On note d'abord que CS_{pr} est un système M-compatible fermé. D'après la Proposition 6.4, il existe une D-réduction minimale associée à un système M-compatible fermé minimal dont les éléments sont des classes premières. D'après le Corollaire 6.2, cette D-réduction est une sous-machine minimale de $A_{CS_{pr}}$. ❐

6.2 Construction d'une D-réduction minimale

Selon les précédantes considérations, le problème de la détermination d'une réduction minimale déterministe à partir d'une machine observable complètement spécifiée non déterministe A peut être résolu par l'une des deux méthodes suivantes :

- La détermination d'une D-sous-machine minimale de $A_{CS_{pr}}$ ayant pour état initial $s_0 \in C_0$.

ou

- La construction d'une machine de Mealy déterministe correspondant à un système M-compatible fermé minimal $CS \subseteq CS_{pr}$.

Les deux méthodes précédantes nécessitent la construction de l'ensemble de toutes les classes premières de la machine de Mealy observable complètement spécifiée non déterministe A. Pour cela, nous devons d'abord trouver toutes les classes M-compatible maximales. Par la suite, nous devons construire l'ensemble des classes premières CS_{pr} en adaptant la méthode dans [Grasselli 65]. En dernier lieu, pour la première méthode nous

devons construire la machine de Mealy complètement spécifiée non déterministe $A_{CS_{pr}}$ et choisir une D-sous-machine minimale de $A_{CS_{pr}}$ ayant pour état initial $s_0 \in C_0$, par contre pour la deuxième méthode nous devons utiliser la méthode des équations booléennes [Grasselli 65] pour trouver un système M-compatible fermé minimal $CS \subseteq CS_{pr}$.

Pour construire les classes M-compatible maximales, on peut utiliser l'une des deux approches suivantes.

Première approche

On construit de façon itérative les classes M-compatible. Un sous ensemble C d'états de A est dit *1-M-compatible* si $\bigcap_{s \in C} h^2(s, x) \neq \varnothing$ pour tout $x \in X$.

Étape1 : Trouver toutes les classes maximales 1-M-compatible. On note CS_0 l'ensemble de ces classes. On pose $I := 0$.

Étape2 : On considère chaque classe C de l'ensemble CS_I. Si pour tout $x \in X$ il existe $y \in \bigcap_{s \in C} h^2(s, x)$ et $C_j \in CS_I$ tels que $\mathrm{NEXT}(C, x/y) \subseteq C_j$, alors $C \in CS_{I+1}$. Sinon trouver tous les sous ensembles propres de C et les inclure dans CS_{I+1}. Retirer chaque élément de CS_{I+1} qui est un sous ensemble propre d'un autre élément de CS_{I+1}.

Étape3 : Si $CS_{I+1} \neq CS_I$ alors $I := I+1$ et aller à l'étape2. Sinon CS_I contient toutes les classes M-compatible maximales.

Seconde approche

On adapte la table de compatibilité dans [Kohavi 78] pour obtenir les classes M-compatible de façon similaire à [Damiani 94]. La différence par rapport à la table dans [Grasselli 65] est que chaque fois que, pour une paire d'états, on ne peut reconnaître directement qu'ils sont M-compatibles ou M-incompatibles, la case pour cette paire représente une table où chaque élément correspond à une combinaison d'entrée/sortie, et l'on marque la paire d'états impliquée ou on la laisse vide si cette entrée/sortie n'est pas spécifiée pour chacun des états de la paire.

La table doit ensuite être mise à jour de façon récursive. Dans chaque case où des conditions sont présentes, on simplifie ces conditions jusqu'à obtenir une table qu'on ne peut plus simplifier. Les paires d'états qui n'ont pas de croix dans la case qui leurs est associée sont M-compatible. À partir de la table finale, on peut trouver les classes M-compatible maximales.

Nous illustrons ces deux méthodes à travers l'exemple de la machine de Mealy observable complètement spécifiée non déterministe de la Figure 6.3.

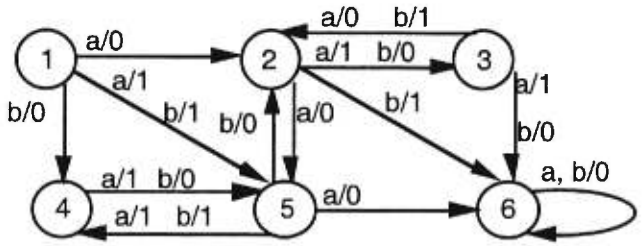


Figure 6.3 : Une machine de Mealy non déterministe

première approche

CS_0={{1, 2, 3, 4, 5}, {1, 2, 3, 5, 6}}.

CS_1={{1, 2, 3, 4}, {1, 2, 3, 5}, {1, 2, 4, 5}, {1, 3, 4, 5}, {2, 3, 4, 5}, {1, 2, 3, 6}, {1, 2, 5, 6}, {1, 3, 5, 6}, {2, 3, 5, 6}}.

CS_2={{1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {2, 3, 4}, {1, 2, 5}, {1, 3, 5}, {1, 4, 5}, {2, 4, 5}, {3, 4, 5}, {1, 2, 6}, {1, 3, 6}, {1, 5, 6}, {2, 3, 5, 6}}.

CS_3={{1, 2, 3}, {1, 2, 4}, {2, 3, 4}, {1, 2, 5}, {1, 3, 5}, {1, 4, 5}, {2, 4, 5}, {2, 3, 5, 6}, {1, 6}}.

CS_4={{1, 2, 3}, {2, 3, 4}, {1, 2, 5}, {1, 3, 5}, {1, 4, 5}, {2, 3, 5, 6}}.

CS_5={{1, 4}, {4, 5}, {1, 2, 3}, {1, 2, 5}, {2, 3, 4}, {2, 3, 5, 6}}=CS_6.

Les classes M-compatible maximales sont :

{1, 4}, {4, 5}, {1, 2, 3}, {1, 2, 5}, {2, 3, 4}, et {2, 3, 5, 6}.

Seconde approche

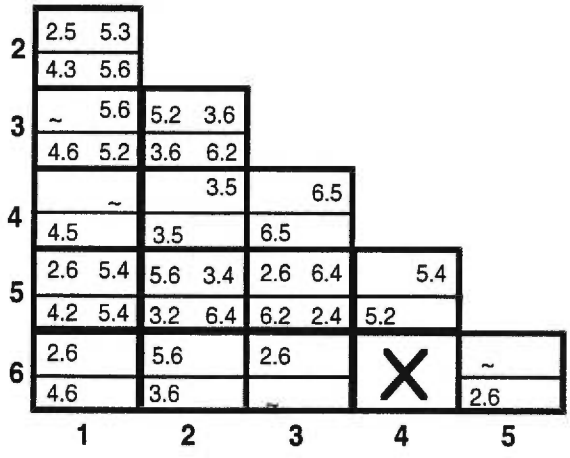


Figure 6.4 : La table de compatibilité à la première étape

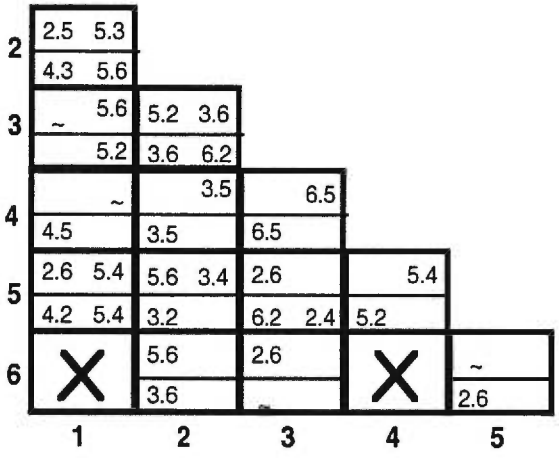


Figure 6.5 : La table de compatibilité à la dernière étape

Les figures 6.4 et 6.5 illustrent la table de compatibilité à différentes étapes. À partir

de la dernière table, on déduit les classes M-compatible maximales.

L'ensemble des classes premières est CS_{pr}={{1}, {2}, {3}, {4}, {5}, {6}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {2, 3}, {2, 4}, {2, 5}, {2, 6}, {3, 4}, {4, 5}, {1, 2, 3}, {1, 2, 5}, {2, 3, 4}, {2, 3, 6}, {2, 5, 6}, {3, 5, 6}, {2, 3, 5, 6}}.

Sélection d'une D-sous-machine minimale de $A_{CS_{pr}}$.

Les classes dans CS_{pr} sont numérotées de 1 à 23 et représentent les états de la machine $A_{CS_{pr}}$ (Figure 6.6).

État \ entrée/sortie	a/0	a/1	b/0	b/1
1	2, 7, 11, 12, 13, 14, 17, 18, 19, 20, 21, 23	5, 10, 13, 16, 18, 21, 22, 23	4, 9, 12, 15, 16, 19	5, 10, 13, 16, 18, 21, 22, 23
2	5, 10, 13, 16, 18, 21, 22, 23	3, 8, 11, 15, 17, 19, 20, 22, 23	3, 8, 11, 15, 17, 19, 20, 22, 23	6, 14, 20, 21, 22, 23
3	2, 7, 11, 12, 13, 14, 17, 18, 19, 20, 21, 23	6, 14, 20, 21, 22, 23	6, 14, 20, 21, 22, 23	2, 7, 11, 12, 13, 14, 17, 18, 19, 20, 21, 23
4		5, 10, 13, 16, 18, 21, 22, 23	5, 10, 13, 16, 18, 21, 22, 23	
5	6, 14, 20, 21, 22, 23	4, 9, 12, 15, 16, 19	2, 7, 11, 12, 13, 14, 17, 18, 19, 20, 21, 23	4, 9, 12, 15, 16, 19
6	6, 14, 20, 21, 22, 23		6, 14, 20, 21, 22, 23	
7	13, 18, 21, 23	22, 23	15, 19	22, 23
8	2, 7, 11, 12, 13, 14, 17, 18, 19, 20, 21, 23	22, 23		13, 18, 21, 23
9		5, 10, 13, 16, 18, 21, 22, 23	16	
10	14, 20, 21, 23	16	12, 19	16
11	13, 18, 21, 23	22, 23	22, 23	14, 20, 21, 23
12		22, 23	22, 23	
13	22, 23	15, 19	11, 17, 19, 20, 23	
14	22, 23		22, 23	
15		22, 23	22, 23	
16		16	13, 18, 21, 23	
17	13, 18, 21, 23	22, 23		21, 23
18	21, 23		19	
19		22, 23	22, 23	
20	21, 23		22, 23	
21	22, 23		20, 23	
22	14, 20, 21, 23		14, 20, 21, 23	
23	21, 23		20, 23	

Figure 6.6 : La table de transition de la machine $A_{CS_{pr}}$

La figure 6.7 illustre une D-sous-machine minimale de A_{CSpr} où l'état 17 représentant la classe première {1, 2, 3} est choisit comme état initial.

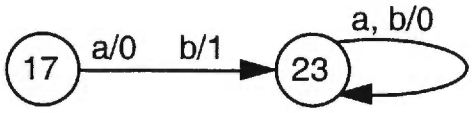


Figure 6.7 : Une D-sous-machine minimale de A_{CSpr}

La méthode des équations booléennes.[Pyne 61]

Cette méthode utilise les techniques d'optimisation d'une fonction sous des contraintes linéaires. On cherche à minimiser le nombre de classes premières choisies pour construire un système M-compatible fermé.

$$\text{Minimiser} \sum_{i=1}^{i=23} c_i$$

Sous les contraintes

$c_1+c_7+c_8+c_9+c_{10}+c_{17}+c_{18}=1$

$\overline{c_1}+(c_2+c_5+c_7+c_{10}+c_{11}+c_{12}+c_{13}+c_{14}+c_{16}+c_{17}+c_{18}+c_{19}+c_{20}+c_{21}+c_{22}+c_{23})$
$(c_4+c_5+c_9+c_9+c_{10}+c_{12}+c_{13}+c_{15}+c_{16}+c_{18}+c_{19}+c_{21}+c_{22}+c_{23})=1$

$\overline{c_2}+(c_3+c_5+c_8+c_{10}+c_{11}+c_{13}+c_{15}+c_{16}+c_{17}+c_{18}+c_{19}+c_{20}+c_{21}+c_{22}+c_{23})$
$(c_3+c_6+c_8+c_{11}+c_{14}+c_{15}+c_{17}+c_{19}+c_{20}+c_{21}+c_{22}+c_{23})=1$

$\overline{c_3}+(c_2+c_6+c_7+c_{11}+c_{12}+c_{13}+c_{14}+c_{17}+c_{18}+c_{19}+c_{20}+c_{21}+c_{22}+c_{23})$
$(c_2+c_6+c_7+c_{11}+c_{12}+c_{13}+c_{14}+c_{17}+c_{18}+c_{19}+c_{20}+c_{21}+c_{22}+c_{23})=1$

$\overline{c_4}+(c_5+c_{10}+c_{13}+c_{16}+c_{18}+c_{21}+c_{22}+c_{23})(c_5+c_{10}+c_{13}+c_{16}+c_{18}+c_{21}+c_{22}+c_{23})=1$

$\overline{c_5}+(c_4+c_6+c_9+c_{12}+c_{14}+c_{15}+c_{16}+c_{19}+c_{20}+c_{21}+c_{22}+c_{23})$
$(c_2+c_4+c_7+c_9+c_{11}+c_{12}+c_{13}+c_{14}+c_{15}+c_{16}+c_{17}+c_{18}+c_{19}+c_{20}+c_{21}+c_{23})=1$

$\overline{c_6}+(c_6+c_{14}+c_{20}+c_{21}+c_{22}+c_{23})(c_6+c_{14}+c_{20}+c_{21}+c_{22}+c_{23})=1$

$\overline{c_7}+(c_{13}+c_{18}+c_{21}+c_{22}+c_{23})(c_{15}+c_{19}+c_{22}+c_{23})=1$

$\overline{c_8}+(c_2+c_7+c_{11}+c_{12}+c_{13}+c_{14}+c_{17}+c_{18}+c_{19}+c_{20}+c_{21}+c_{22}+c_{23})(c_{13}+c_{18}+c_{21}+c_{23})=1$

$\overline{c_9}+(c_5+c_{10}+c_{13}+c_{16}+c_{18}+c_{21}+c_{22}+c_{23})(c_{16})=1$

$\overline{c_{10}}+(c_{14}+c_{16}+c_{20}+c_{21}+c_{23})(c_{12}+c_{16}+c_{19})=1$

$\overline{c_{11}}+(c_{13}+c_{18}+c_{21}+c_{22}+c_{23})(c_{14}+c_{20}+c_{21}+c_{22}+c_{23})=1$

$\overline{c_{12}}+(c_{22}+c_{23})(c_{22}+c_{23})=1$

$\overline{c_{13}}+(c_{15}+c_{19}+c_{22}+c_{23})(c_{11}+c_{17}+c_{19}+c_{20}+c_{23})=1$

$\overline{c_{14}}+(c_{22}+c_{23})(c_{22}+c_{23})=1$

$\overline{c_{15}}+(c_{22}+c_{23})(c_{22}+c_{23})=1$

$\overline{c_{16}}+(c_{16})(c_{13}+c_{18}+c_{21}+c_{23})=1$

$\overline{c_{17}}+(c_{13}+c_{18}+c_{21}+c_{22}+c_{23})(c_{21}+c_{23})=1$

$\overline{c_{18}}+(c_{21}+c_{23})(c_{19})=1$

$\overline{c_{19}}+(c_{22}+c_{23})(c_{22}+c_{23})=1$

$\overline{c_{20}}+(c_{21}+c_{23})(c_{22}+c_{23})=1$

$\overline{c_{21}}+(c_{22}+c_{23})(c_{20}+c_{23})=1$

$\overline{c_{22}}+(c_{14}+c_{20}+c_{21}+c_{23})(c_{14}+c_{20}+c_{21}+c_{23})=1$

$\overline{c_{23}}+(c_{21}+c_{23})(c_{20}+c_{23})=1$

La première équation impose la sélection d'au moins une classe première contenant l'état initial de notre machine de départ. Les autres équations imposent la sélection d'un ensemble impliqué chaque fois que la classe correspondante est sélectionnée.

Une solution pour ce système est $c_{17}=c_{23}=1$. Elle permet l'obtention de la même D-réduction (Figure 6.7) de la machine (Figure 6.3).

6.3 Conclusion

Nous avons décrit dans ce travail une méthode pour construire à partir d'une spécification non déterministe une implantation déterministe minimale dont les traces forment un sous ensemble des traces de la spécification (l'implantation est une réduction de la spécification). Nous avons aussi montré que le précédant travail sur ce problème [Damiani 94] est erroné. Nos résultats peuvent être utilisés dans la conception des systèmes distribués et concurrents; en particulier, lorsqu'une spécification pour un sous module dans une conception hiérarchique est obtenue sous la forme d'une machine non déterministe. Le travail décrit permet d'obtenir à partir de cette dernière une solution déterministe ayant un nombre minimal d'états pour l'utiliser comme spécification pour une implantation. Le problème avec l'approche considérée est sa complexité, la détermination d'une réduction déterministe minimale à partir d'une machine non déterministe est au moins aussi complexe que la minimisation des états d'une machine déterministe partiellement spécifiée, qui est exponentielle en fonction du nombre des états [Pfleeger 73]. Pour travailler avec des spécifications non déterministes de taille réelle, on a besoin d'un outil utilisant une procédure heuristique qui peut être obtenu à partir de l'approche décrite. Le travail [Kam 95] décrit une approche basée sur des techniques implicites pour la minimisation des états pour différentes classes de machines de Mealy. Un outil basé sur l'approche a été développé au sein du groupe de conception assistée par ordinateur (CAD) de l'université de Californie à Berkeley.

Il serait aussi intéressant de développer une autre technique pour transformer une spécification non déterministe en une implantation déterministe en prenant en compte la testabilité de cette dernière, car une implantation avec un nombre minimal d'états n'est pas nécessairement la plus facilement testable.

CHAPITRE 7

LA CONSTRUCTION DANS LE MODÈLE DES AUTOMATES À ENTRÉES ET SORTIES

Une première approche pour la généralisation du travail du chapitre 5 consistait à procéder étape par étape. La première étape étant la résolution du problème de construction de sous-modules dans le modèle des machines de Mealy non déterministes complètement spécifiées. Le travail de Petrenko et Yevtushenko [Petrenko 98] se situe dans cette approche. Il présente de plus une approche pour traiter les cycles étiquetés seulement par des actions internes. Le cas des machines de Mealy partiellement spécifiées restait ouvert dans ce travail.

De nôtre coté, nous avons privilégié une autre approche [Drissi 99a]. Dans un premier temps, nous avons généralisé le modèle en utilisant celui des automates à entrées et sorties. Afin de pouvoir considérer une machine de Mealy comme un cas particulier de notre modèle nous avons opté pour des automates à entrées et sorties partiellement spécifiés. Un automate à entrées et sorties observe toutes les actions présentes dans son alphabet mais ne contrôle que les actions produites par lui, c'est-à-dire ses sorties. Dans le travail sur les machines de Mealy, l'ensemble des entrées du module à concevoir est défini de façon implicite. C'est la réunion des sorties internes du contexte et des actions reçues par le système à partir de l'environnement ne faisant pas partie des entrées du contexte. Dans un deuxième temps, nous avons opté pour une définition explicite de cet ensemble afin de permettre au module à concevoir d'observer certaines interaction du contexte avec l'environnement. Le problème de construction de sous-modules consiste dans ce cas à résoudre l'équation $(C||X)\Re A$ sous la contrainte $I_X=In$ où In est un ensemble d'actions vérifiant $(I_A\backslash I_C)\cup(O_C\backslash O_A)\subseteq In\subseteq I_A\cup O_C$. Dans le cas où $In=I_A\cup O_C$, le module à concevoir observe toutes les actions présentes dans le système. Par contre, si $In=(I_A\backslash I_C)\cup(O_C\backslash O_A)$, nous avons l'observabilité minimale pour le module à concevoir. C'est ce dernier cas qui était

considéré pour les machines de Mealy. Il faut noter à ce niveau que l'existence d'une solution dépend du degré d'observation alloué au module à concevoir. Au lieu de résoudre le problème de construction de sous-modules pour chacune des relations de conformité trace équivalence, réduction et quasi-équivalence, nous avons dans un troisième temps défini deux relations de conformité (voir chapitre 3). La première relation de conformité, la réalisation sécuritaire, capture ce qui est commun aux trois relations de conformité précédantes. La deuxième nouvelle relation de conformité, l'implantation conforme, est inspirée de la relation sous-type présente dans les langages orienté-objet. C'est une relation de conformité générique dans le sens où chacune des relations de conformité trace équivalence, réduction et quasi-équivalence peut être considérée comme un cas particulier de la relation d'implantation conforme.

7.1 La conception de sous-module

7.1.1 L'architecture

Nous utilisons l'architecture illustrée dans la figure 7.1 afin de discuter le problème de la conception d'une composante d'un système composé.

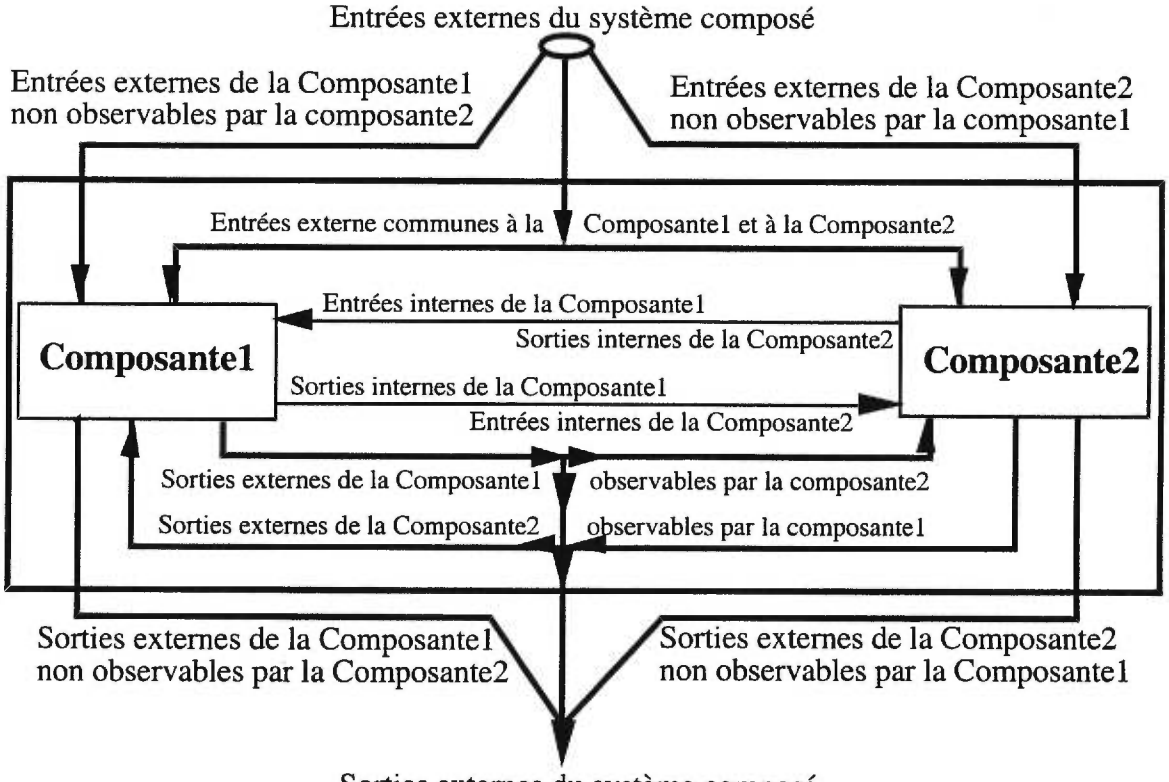


Figure 7.1 : L'architecture de composition

Nous considérons la classe de systèmes qui peuvent être modélisés par deux automates à entrées et sorties déterministes qui communiquent entre eux et avec l'environnement. L'ensemble des entrées externes du système composé soumises par l'environnement est partitionné en trois sous ensembles disjoints. Le premier sous ensemble représente les entrées externes de la Composante1 non observables par la Composante2. Le second sous ensemble représente les entrées externes communes aux deux composantes. Le troisième sous ensemble représente les entrées externes de la Composante2 non observables par la Composante1. De façon analogue, l'ensemble des sorties externes délivrées par le système composé à l'environnement est partitionné en quatres sous ensembles disjoints. Le premier sous ensemble représente les sorties externes de la Composante1 non observables par la Composante2. Le second sous ensemble représente les sorties externes de la Composante1 observables par la Composante2. Le troisième sous ensemble représente les sorties externes de la Composante2 non observables par la Composante1. Enfin, le quatrième ensemble représente les sorties externes de la Composante2 observables par la Composante1. De plus les sorties interne d'une composante, non observable par l'environnement, seront des entrées internes pour l'autre composante.

7.1.2 Le problème

Dans ce chapitre, nous nous intéressons à la résolution de l'équation $(C\|X)\Re A$ sous la contrainte $I_X=In$ dans le modèle des automates à entrées et sorties. Le premier automate à entrées et sorties déterministe, appelé le contexte C, modélise la partie connue du système, c'est-à-dire, celle qui existe au départ. Le deuxième automate à entrées et sorties déterministe A, modélise le système complet désiré. L'ensemble In donné vérifiant $(I_A \setminus I_C) \cup (O_C \setminus O_A) \subseteq In \subseteq I_A \cup O_C$ décrit l'ensemble des entrées du module à concevoir X.

Nous proposons dans ce chapitre deux algorithmes et nous prouvons qu'ils sont corrects. Le premier algorithme permet de générer la solution générique de l'équation $(C\|X)\Re A$ sous la contrainte $I_X=In$, notée $Sol_{\mathfrak{S}}$, lorsque la relation de conformité utilisée est la réalisation sécuritaire. Le deuxième algorithme permet de générer la solution générique de l'équation $(C\|X)\Re A$ sous la contrainte $I_X=In$, notée Sol, lorsque la relation de conformité utilisée est l'implantation conforme. De plus, dans chacun des cas nous caractérisons l'ensemble des solutions possibles de l'équation.

L'existence d'une solution pour une équation donnée dépends de l'ensemble des entrées requis pour le module à concevoir.

Lemme 7.1 : Si pour un ensemble *In* donné il n'existe pas de solution, alors pour tout sous ensemble de *In* il n'existe pas de solution.

7.2 La solution pour la relation de conformité réalisation sécuritaire

7.2.1 La solution sécuritaire générique

Pour décrire de façon intuitive la méthode, nous utilisons un automate à entrées et sorties chaotique qui représente toutes les traces sur l'alphabet d'entrées *In* et sur l'alphabet de sorties $(I_C \backslash I_A) \cup (O_A \backslash O_C)$. Cette automate chaotique peut être défini formellement par $Ch=(S_{Ch}, I_{Ch}, O_{Ch}, T_{Ch}, s_{oCh})$ où $S_{Ch}=\{ch\}$, $I_{Ch}=In$, $O_{Ch}=(I_C \backslash I_A) \cup (O_A \backslash O_C)$, $s_{oCh}=ch$ et $T_{Ch}=\{(ch, a, ch) \mid a \in I_{Ch} \cup O_{Ch}\}$. Toute solution pour notre problème a un ensemble de traces inclus dans l'ensemble des traces de l'automate chaotique. L'idée principale de la méthode consiste en l'élimination de toute trace contenue dans l'automate chaotique, dont la combinaison avec des traces du contexte *C* dans l'environnement associé par réflexion à la spécification du système *A* décrit au moins un comportement non conforme. Ceci permettra de capturer l'ensemble des traces permises (s'il n'est pas vide) de la composante à concevoir sous la forme d'un automate à entrées et sorties $Sol_{\S}$, appelé la solution sécuritaire générique. Une trace permise est une trace d'un automate à entrées et sorties qui est une solution de l'équation $(C \parallel X) \leq_{\S} A$ sous la contrainte $I_X=In$.

Afin de déterminer les traces permises sous la forme d'un automate à entrées et sorties, nous utiliserons Algorithme 7.1 dont le pseudo-code est présenté dans le paragraphe suivant. Cet algorithme requiert en entrée un automate à entrées et sorties déterministe *C* représentant le comportement du contexte existant , un automate à entrées et sorties déterministe *A* représentant le comportement souhaité du système complet et un ensemble *In* vérifiant $(I_A \backslash I_C) \cup (O_C \backslash O_A) \subseteq In \subseteq I_A \cup O_C$ qui représente l'ensemble des entrées désiré pour la composante à concevoir. L'algorithme se compose de trois étapes. Dans la première étape, nous construisons les formes complétées de *C* et de la réflexion $\tilde{A}$ de *A*, puis la composition $R=Ief(C)\|Ch\|Ief(\tilde{A})$. Puisque un état de *R* est un triplet contenant un état de chacun des automates $Ief(C)$, *Ch* et $Ief(\tilde{A})$, nous remplacerons cet état par $Fail_R$ chaque fois qu'il contient $Fail_{Ief(\tilde{A})}$ ou $Fail_{Ief(C)}$. La complexité dans le pire cas de cette étape est polynomiale en fonction du nombre d'états de *C*, du nombre d'états de *A* et du nombre d'éléments dans l'alphabet de $A \| C$. Nous pouvons remarquer à ce niveau que l'introduction de l'automate chaotique relevait surtout d'un besoin didactique que d'un besoin intrinsèque de l'algorithme vue que $Ief(C)\|Ch\|Ief(\tilde{A})=Ief(C)\|Ief(\tilde{A})$. Dans la deuxième étape, nous remplaçons dans *R*

toutes les actions dont les étiquettes ne font pas partie de l'alphabet de la composante à concevoir par l'action interne τ, ensuite nous déterminons l'automate à entrées et sorties déterministe R_1 tel que $Tr_{R_1}=Tr_R$. Puisque un état de R_1 correspond à un sous ensemble d'états de R, nous remplaçons tout état de R_1 qui contient $Fail_R$ par $Fail_{R_1}$. Nous notons $R_1=H_{(I_A \cup O_C)\setminus In)}(R)$ la transformation réalisée à l'étape 2. La complexité dans le pire cas de cette étape est exponentielle en fonction du nombre d'états de C et de A. Dans la troisième et dernière étape, nous éliminons de façon récursive toutes les transitions présentent dans R_1 qui mènent à l'état $Fail_{R_1}$. La complexité dans le pire cas de cette étape est polynomiale en fonction du nombre d'états de R_1.

Algorithme 7.1

Entrées : La spécification du contexte C, la spécification du système A et un ensemble d'entrées pour la solution *In*.

Sortie : un automate à entrées et sorties $Sol_{\S}$ tel que $I_{Sol_{\S}}=In$ (égale à R_1) qui satisfait l'équation $(C \| Sol_{\S}) \leq_{\S} A$ lorsqu'une solution existe.

Étape 1 : $R:=Ief(C) \| Ch \| Ief(\tilde{A})$.

Étape 2 : $R_1:=H_{(I_A \cup O_C)\setminus In)}(R)$.

Étape 3 : (*Élimination des traces non conformes*)

TANT QUE il existe une transition s - $t \rightarrow Fail_{R_1}$ **FAIRE**

SI $t \in (I_C \setminus I_A) \cup (O_A \setminus O_C)$ **ALORS** éliminer cette transition;

SINON

SI $s=s_{0R_1}$ **ALORS** retourner "PAS DE SOLUTION"; **STOP**;

SINON Remplacer chaque transition c- $t' \rightarrow s$ par c- $t' \rightarrow Fail_{R_1}$;

$R_1:=CC(R_1)$;

Théorème 7.1 : Étant donnés deux automates à entrées et sorties déterministes A et C, et étant donné un ensemble *In* tel que $(I_A \setminus I_C) \cup (O_C \setminus O_A) \subseteq In \subseteq I_A \cup O_C$, si Algorithme 7.1 produit un automate à entrées et sorties $Sol_{\S}$ alors $(C \| Sol_{\S}) \leq_{\S} A$ avec $I_{Sol_{\S}}=In$, sinon il n'existe aucune solution pour l'équation $(C \| X) \leq_{\S} A$ ayant *In* comme ensemble d'entrées.

Preuve du Théorème 7.1 :

Première partie :

Si Algorithme 7.1 produit un automate à entrées et sorties $Sol_{\S}$ alors on doit montrer que $(C \| Sol_{\S}) \leq_{\S} A$. Supposons qu'il existe $\sigma \in (I_{(C \| Sol_{\S} \| \tilde{A})} \cup O_{(C \| Sol_{\S} \| \tilde{A})})^*$ telle que :

$Pr_C(\sigma) \in Tr_C.(I_C \cup \{\varepsilon\}) \wedge Pr_{Sol_{\S}}(\sigma) \in Tr_{Sol_{\S}}.(I_{Sol_{\S}} \cup \{\varepsilon\}) \wedge Pr_A(\sigma) \in Tr_A.(O_A \cup \{\varepsilon\})$

$\wedge \sigma \notin Tr_{(C \| Sol_{\S} \| \tilde{A})}$

nous considérons une trace σ pour laquelle la propriété précédante n'est vérifiée pour aucun de ses préfixes propres.

Puisque : $Pr_C(\sigma)\in Tr_{Ief(C)} \wedge Pr_{Sol_{\S}}(\sigma)\in Tr_{Ch} \wedge Pr_A(\sigma)\in Tr_{Ief(\tilde{A})}$ alors $\sigma\in Tr_R$

ceci implique que σ menait à l'état *Fail* à une certaine itération de Algorithme 7.1 et par conséquent a été éliminée. Posons $\sigma=\sigma'.t$,

Si $t\in O_{Sol_{\S}}$ alors $Pr_{Sol_{\S}}(\sigma)\notin Tr_{Sol_{\S}}(I_{Sol_{\S}}\cup\{\varepsilon\})$, contradiction.

Si $t\in (I_A\cup O_C)$ alors après l'élimination de t, σ' mènerait à l'état $Fail$ et donc $\sigma'\notin Tr_{(C\|Sol_{\S}\|\tilde{A})}$, contradiction.

Nous concluons que $(C\|Sol_{\S})\leq_{\S}A$.

Deuxième partie :

Si Algorithme 7.1 retourne "PAS DE SOLUTION", alors il existe $\sigma\in(I_A\cup O_C)^*$, avec $\sigma=\sigma'.t$ et $Pr_C(\sigma')\in Tr_C$ et $Pr_A(\sigma')\in Tr_A$ et $t\in (I_A\cap I_C)\cup(O_A\cap O_C)$, telle que $(s_{0R})_{\sigma}=Fail_R$.

Supposons qu'il existe un automate à entrées et sorties B tel que $I_B=In$ et $(C\|B)\leq_{\S}A$.

Si $\sigma'=\varepsilon$ alors $\sigma'\in Tr_{(C\|B\|\tilde{A})}$, sinon par induction nous montrons que $\sigma'\in Tr_{(C\|B\|\tilde{A})}$, posons $|\sigma'|=n$ et notons $\sigma'_{[k]}$ le préfixe de σ' de longueur k pour $0\leq k\leq n$,

- Puisque $(C\|B)\leq_{\S}A$, on a $Pr_B(\sigma'_{[1]})\in Tr_B.(I_B\cup\{\varepsilon\}) \Rightarrow \sigma'_{[1]}\in Tr_{(C\|B\|\tilde{A})}$,
- Supposons que $\sigma'_{[k]}\in Tr_{(C\|B\|\tilde{A})}$ pour $1\leq k<n$, et montrons que $\sigma'_{[k+1]}\in Tr_{(C\|B\|\tilde{A})}$

Puisque $\sigma'_{[k]}\in Tr_{(C\|B\|A)}$ alors $Pr_B(\sigma'_{[k]})\in Tr_B$, et donc $Pr_B(\sigma'_{[k+1]})\in Tr_B.(I_B\cup\{\varepsilon\})$

Puisque $(C\|B)\leq_{\S}A$, on a $Pr_B(\sigma'_{[k+1]})\in Tr_B.(I_B\cup\{\varepsilon\}) \Rightarrow \sigma'_{[k+1]}\in Tr_{(C\|B\|\tilde{A})}$.

D'après le principe d'induction $\sigma'\in Tr_{(C\|B\|\tilde{A})}$.

Maintenant,

Si $t\in (I_A\cap I_C)$ alors $Pr_C(\sigma)\in Tr_C.(I_C\cup\{\varepsilon\}) \wedge Pr_B(\sigma)\in Tr_B.(I_B\cup\{\varepsilon\}) \wedge Pr_A(\sigma)\in Tr_A$

Mais $\sigma\notin Tr_{(C\|B\|\tilde{A})}$ car $Pr_C(\sigma)\notin Tr_C$, contradiction.

Si $t\in (O_A\cap O_C)$ alors $Pr_C(\sigma)\in Tr_C \wedge Pr_B(\sigma)\in Tr_B.(I_B\cup\{\varepsilon\}) \wedge Pr_A(\sigma)\in Tr_A.(O_A\cup\{\varepsilon\})$

Mais $\sigma\notin Tr_{(C\|B\|\tilde{A})}$ car $Pr_A(\sigma)\notin Tr_A$, contradiction.

Nous concluons qu'il n'existe pas d'automate à entrées et sorties B tel que $I_B=In$ et $(C\|B)\leq_{\S}A$. □

Exemple :

Nous considérons les automates à entrées et sorties A et C de la Figure 7.2. L'automate à entrées et sorties C décrit la spécification du contexte avec $I_C=\{x_1, x_2, z_1, z_2, z_3, z_4\}$ et $O_C=\{u, y_1, y_2\}$. L'automate à entrées et sorties A décrit la spécification du système complet désiré avec $I_A=\{x_1, x_2, x_3\}$ et $O_A=\{y_1, y_2, y_3\}$. L'ensemble des entrées requis pour le module à concevoir est $In=\{x_1, x_3, u\}$. Après application de l'algorithme 7.1,

nous obtenons la solution $Sol_{\S}$ ayant pour ensemble de sorties $O_{Sol_{\S}}=\{z_1, z_2, z_3, z_4, y_3\}$.

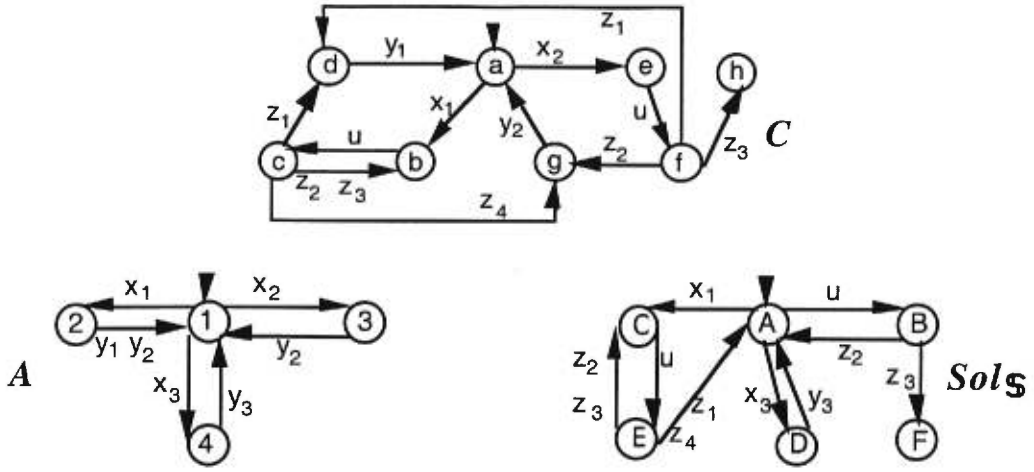


Figure 7.2 : Les automates à entrées et sorties C, A et $Sol_{\S}$

7.2.2 L'ensemble des solutions

La solution obtenue par la méthode décrite précédemment est générique. On peut dériver à partir d'elle l'ensemble des solutions pour l'équation $(C \parallel X) \leq_{\S} A$ sous la contrainte $I_X=In$.

Théorème 7.2 : Étant donnés deux automates à entrées et sorties déterministes A et C, et étant donné un ensemble In tel que $(I_A \setminus I_C) \cup (O_C \setminus O_A) \subseteq In \subseteq I_A \cup O_C$; un automate à entrées et sorties B, avec $I_B=In$ et $O_B=O_{Sol_{\S}}$, est une solution de l'équation $(C \parallel X) \leq_{\S} A$ si et seulement si $B \leq_{\S} Sol_{\S}$.

Preuve du Théorème 7.2 :

Nous utiliserons dans cette preuve que $(C \| Sol_{\S}) \leq_{\S} A$, c'est-à-dire pour toute trace

$\sigma \in (I_{(C \| Sol_{\S} \| \tilde{A})} \cup O_{(C \| Sol_{\S} \| \tilde{A})})^*$

$Pr_C(\sigma) \in Tr_C.(I_C \cup \{\varepsilon\}) \wedge Pr_{Sol_{\S}}(\sigma) \in Tr_{Sol_{\S}}.(I_{Sol_{\S}} \cup \{\varepsilon\}) \wedge Pr_A(\sigma) \in Tr_A.(O_A \cup \{\varepsilon\}) \Rightarrow$

$\sigma \in Tr_{(C \| Sol_{\S} \| \tilde{A})}$

Première partie : $(C \| B) \leq_{\S} A \Rightarrow B \leq_{\S} Sol_{\S}$

Considérons $\sigma \in (I_{Sol_{\S}} \cup O_{Sol_{\S}})^*$, nous devons montrer que :

$$\sigma \in Tr_{Sol_{\S}}.(O_{Sol_{\S}} \cup \{\varepsilon\}) \wedge \sigma \in Tr_B.(I_B \cup \{\varepsilon\}) \Rightarrow \sigma \in Tr_{(B \| \widetilde{Sol_{\S}})}$$

Posons $\sigma=\sigma_1.t$.

Cas 1 : $t \in I_{Sol_{\S}}$,

Nous avons $\sigma \in Tr_{Sol_{\S}} \wedge \sigma \in Tr_B.I_B$,

Par construction de $Sol_{\S}$, il existe $\sigma_2.t \in Tr_{(C\|\tilde{A})}$ telle que $Pr_{Sol_{\S}}(\sigma_2.t)=\sigma$

De plus $Pr_C(\sigma_2.t)\in Tr_C \wedge Pr_A(\sigma_2.t)\in Tr_A \wedge Pr_B(\sigma_2.t)\in Tr_B.I_B$

Puisque $(C\|B)\leq_{\S}A$ alors $\sigma\in Tr_B$

Donc $\sigma\in Tr_{(B\|\widetilde{Sol_{\S}})}$.

Cas 2 : $t\in O_{Sol_{\S}}$

Nous avons $\sigma\in Tr_{Sol_{\S}}.O_{Sol_{\S}} \wedge \sigma\in Tr_B$,

Si $\sigma_1\neq\varepsilon$, par construction de $Sol_{\S}$, il existe $\sigma_3\in Tr_{(C\|\tilde{A})}$ telle que $Pr_{Sol_{\S}}(\sigma_3)=\sigma_1$ et $Pr_{Sol_{\S}}(\sigma_{3[k]})\neq\sigma_1$ pour $0\leq k<|\sigma_3|$, sinon nous poserons $\sigma_3=\varepsilon$

De plus $Pr_C(\sigma_3.t)\in Tr_C.(I_C\cup\{\varepsilon\})\wedge Pr_A(\sigma_3.t)\in Tr_A.(O_A\cup\{\varepsilon\}) \wedge Pr_B(\sigma_3.t)\in Tr_B$

Puisque $(C\|B)\leq_{\S}A$ alors $\sigma_3.t\in Tr_{(C\|\tilde{A})}$

Si $Pr_B(\sigma_3.t)\notin Tr_{Sol_{\S}}$, par construction de $Sol_{\S}$ il existe

$\sigma_3.\sigma_4.t.\sigma_5.t'\in Tr_{(Ief(C)\|Ief(\tilde{A}))}$ qui mène à l'état *Fail* avec $\sigma_4\in((I_A\cup O_C)\backslash In)^*$,

$\sigma_5\in(I_A\cup O_C)^*$ et $t'\in((I_A\cap I_C)\cup(O_A\cap O_C))$

Puisque $(C\|B)\leq_{\S}A$ alors $Pr_B(\sigma_3.\sigma_4.t.\sigma_5)\in Tr_B$

Maintenant, si $t'\in(I_A\cap I_C)$ alors

$$Pr_C(\sigma_3.\sigma_4.t.\sigma_5.t')\in Tr_C.(I_C\cup\{\varepsilon\})\wedge Pr_B(\sigma_3.\sigma_4.t.\sigma_5.t')\in Tr_B.(I_B\cup\{\varepsilon\})$$
$$\wedge Pr_A(\sigma_3.\sigma_4.t.\sigma_5.t')\in Tr_A$$

Mais $\sigma_3.\sigma_4.t.\sigma_5.t'\notin Tr_{(C\|B\|\tilde{A})}$ car $Pr_C(\sigma_3.\sigma_4.t.\sigma_5.t')\notin Tr_C$, contradiction.

Si $t'\in(O_A\cap O_C)$ alors

$$Pr_C(\sigma_3.\sigma_4.t.\sigma_5.t')\in Tr_C\wedge Pr_B(\sigma_3.\sigma_4.t.\sigma_5.t')\in Tr_B.(I_B\cup\{\varepsilon\})$$
$$\wedge Pr_A(\sigma_3.\sigma_4.t.\sigma_5.t')\in Tr_A.(O_A\cup\{\varepsilon\})$$

Mais $\sigma_3.\sigma_4.t.\sigma_5.t'\notin Tr_{(C\|B\|\tilde{A})}$ car $Pr_A(\sigma_3.\sigma_4.t.\sigma_5.t')\notin Tr_A$, contradiction.

Donc $Pr_B(\sigma_3.t)\in Tr_{Sol_{\S}}$ et $\sigma\in Tr_{(B\|\widetilde{Sol_{\S}})}$

Nous concluons que $B\leq_{\S}Sol_{\S}$

Deuxième partie : $B\leq_{\S}Sol_{\S} \Rightarrow (C\|B)\leq_{\S}A$

Considérons $\sigma\in(I_{(C\|\tilde{A})}\cup O_{(C\|\tilde{A})})^*$, nous devons montrer que :

$$Pr_C(\sigma)\in Tr_C.(I_C\cup\{\varepsilon\})\wedge Pr_B(\sigma)\in Tr_B.(I_B\cup\{\varepsilon\})\wedge Pr_A(\sigma)\in Tr_A.(O_A\cup\{\varepsilon\})$$
$$\Rightarrow \sigma\in Tr_{(C\|B\|\tilde{A})}$$

Posons $\sigma=\sigma_1.t$, avec $\sigma_1\in Tr_{(C\|B\|\tilde{A})}$

Soit $\sigma'=Pr_B(\sigma_1)$ et $|\sigma'|=n$, nous montrons par induction que $\sigma'\in Tr_{Sol_{\S}}$

Si $\sigma'_{[1]}\in O_{Sol_{\S}}$, puisque $B\leq_{\S}Sol_{\S}$ alors

$$\sigma'_{[1]}\in Tr_{Sol_{\S}}.O_{Sol_{\S}} \wedge \sigma'_{[1]}\in Tr_B \Rightarrow \sigma'_{[1]}\in Tr_{Sol_{\S}}$$

Si $\sigma'_{[1]}\in I_{Sol_{\S}}$, alors il existe un préfixe σ_2 deσ_1 tel que $\sigma_2=\sigma_3.\sigma'_{[1]}$ et $Pr_{Sol_{\S}}(\sigma_2)=\sigma'_{[1]}$,

Puisque $(C\|Sol_{\S})\leq_{\S}A$ alors

$Pr_C(\sigma_2)\in Tr_C \wedge Pr_{Sol_{\S}}(\sigma_2)\in Tr_{Sol_{\S}}.I_{Sol_{\S}} \wedge Pr_A(\sigma_2)\in Tr_A$

$\Rightarrow \sigma'_{[1]}=Pr_{Sol_{\S}}(\sigma_2)\in Tr_{Sol_{\S}}$

Supposons que $\sigma'_{[k]}\in Tr_{Sol_{\S}}$ pour $1\leq k<n$, et posons $\sigma'_{[k+1]}=\sigma'_{[k]}.t'$

Si $t'\in O_{Sol_{\S}}$, puisque $B\leq_{\S} Sol_{\S}$ alors

$\sigma'_{[k+1]}\in Tr_{Sol_{\S}}.O_{Sol_{\S}} \wedge \sigma'_{[k+1]}\in Tr_B \Rightarrow \sigma'_{[k+1]}\in Tr_{Sol_{\S}}$

Si $t'\in I_{Sol_{\S}}$, alors il existe un préfixe σ_4 de σ_1 tel que $\sigma_4=\sigma_5.t'$ et $Pr_{Sol_{\S}}(\sigma_4)=\sigma'_{[k+1]}$,

Puisque $(C\|Sol_{\S})\leq_{\S} A$ alors

$Pr_C(\sigma_4)\in Tr_C \wedge Pr_{Sol_{\S}}(\sigma_4)\in Tr_{Sol_{\S}}.I_{Sol_{\S}} \wedge Pr_A(\sigma_4)\in Tr_A$

$\Rightarrow \sigma'_{[k+1]}=Pr_{Sol_{\S}}(\sigma_4)\in Tr_{Sol_{\S}}$

D'après le principe d'induction $\sigma'\in Tr_{Sol_{\S}}$, donc $\sigma_1\in Tr_{(C\|Sol_{\S}\|\tilde{A})}$.

Cas 1 : $t\in O_{Sol_{\S}}$,

on a $Pr_{Sol_{\S}}(\sigma)\in Tr_B$, et puisque $B\leq_{\S} Sol_{\S}$ alors $Pr_{Sol_{\S}}(\sigma)\in Tr_{Sol_{\S}}$

Puisque $(C\|Sol_{\S})\leq_{\S} A$ alors

$Pr_C(\sigma)\in Tr_C.(I_C\cup\{\varepsilon\})\wedge Pr_{Sol_{\S}}(\sigma)\in Tr_{Sol_{\S}}\wedge Pr_A(\sigma)\in Tr_A.(O_A\cup\{\varepsilon\})$

$\Rightarrow \sigma\in Tr_{(C\|Sol_{\S}\|\tilde{A})}$

Donc $\sigma\in Tr_{(C\|A)}$ ce qui entraîne $\sigma\in Tr_{(C\|B\|\tilde{A})}$

Cas 2 : $t\notin O_{Sol_{\S}}$,

Puisque $(C\|Sol_{\S})\leq_{\S} A$

$Pr_C(\sigma)\in Tr_C.(I_C\cup\{\varepsilon\})\wedge Pr_{Sol_{\S}}(\sigma)\in Tr_{Sol_{\S}}.(I_{Sol_{\S}}\cup\{\varepsilon\})\wedge Pr_A(\sigma)\in Tr_A.(O_A\cup\{\varepsilon\})$

$\Rightarrow \sigma\in Tr_{(C\|Sol_{\S}\|\tilde{A})}$

Si $t\in I_{Sol_{\S}}$ alors $Pr_{Sol_{\S}}(\sigma)=\sigma'.t\in Tr_{Sol_{\S}}$

Puisque $B\leq_{\S} Sol_{\S}$ alors $\sigma'.t\in Tr_B$ ce qui entraîne $\sigma\in Tr_{(C\|B\|\tilde{A})}$

Si $t\notin I_{Sol_{\S}}$ alors $Pr_B(\sigma)=\sigma'\in Tr_B$ ce qui entraîne $\sigma\in Tr_{(C\|B\|\tilde{A})}$

Nous concluons que $(C\|B)\leq_{\S} A$. □

Exemple :

La Figure 7.3 illustre d'autres solutions pour l'exemple précédant.

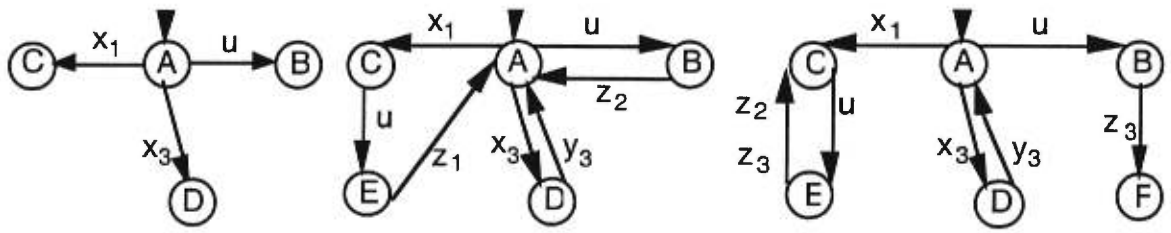


Figure 7.3 : Des réalisations sécuritaires de $Sol_{\S}$

7.3 La solution pour la relation de conformité implantation conforme

Nous nous intéressons maintenant à la résolution de l'équation $(C \| X) \leq_{conf} A$ sous la contrainte $I_X = In$ avec $(I_A \setminus I_C) \cup (O_C \setminus O_A) \subseteq In \subseteq I_A \cup O_C$. La spécification du contexte C est donnée sous la forme d'un automate à entrées et sorties déterministe. La spécification du système complet A est donnée sous la forme d'un automate à entrées et sorties avec traces complètes optionnelles. Afin de simplifier l'algorithme, nous considérons seulement le cas où pour tout état s de l'automate à entrées et sorties IOA_A, $OCT_A(s) = \text{out}(s)$ et $MT_A(s) \subseteq 2^{out(s)}$. Ceci signifie que chaque sortie présente dans s est optionnelle et que au moins une sortie dans chaque élément de MT_A(s) est obligatoire. Un élément de cette classe restreinte est appelé automate à entrées et sorties avec options (*IOAWO* : I/O automaton with options). Nous présentons dans ce qui suit un algorithme pour la construction de la solution générique, notée *Sol*, lorsqu'elle existe. Cette solution générique est obtenu sous la forme d'un automate à entrées et sorties avec traces complètes optionnelles. Par la suite, nous montrons que l'ensemble des solution coïncide avec l'ensemble des implantations conformes de la solution générique.

7.3.1 La solution générique

Nous utilisons l'automate à entrées et sorties $Sol_{\S}$ qui représente la solution générique de l'équation $(C \| X) \leq_{\S} IOA_A$ sous la contrainte $I_X = In$ (voir Section 2) comme point de départ. Toute solution de l'équation $(C \| X) \leq_{conf} A$ sous la contrainte $I_X = In$ a un ensemble de traces inclus dans l'ensemble des traces de $Sol_{\S}$. L'idée principale de notre approche est d'éliminer de $Sol_{\S}$ toute trace dont la combinaison avec certaines traces du contexte C dans l'environnement $\tilde{A}$, peut causer un comportement non conforme par rapport aux traces obligatoires dans MT_A ou aux traces complètes optionnelles dans OCT_A. Ceci permettra de capturer l'ensemble des traces permises (s'il n'est pas vide) de la composante à concevoir sous la forme d'un automate à entrées et sorties avec traces complètes optionnelles *Sol*, appelée la solution générique. Une trace permise est une trace d'un automate à entrées et sorties qui est une solution de l'équation $(C \| X) \leq_{conf} A$ sous la contrainte $I_X = In$.

Afin de déterminer les traces permises sous la forme d'un automate à entrées et sorties avec traces complètes optionnelles, nous utiliserons Algorithme 7.2 dont le pseudo-code sera présenté un peu plus loin. Cet algorithme requiert en entrée un automate à entrées et sorties déterministe C représentant le comportement du contexte existant, un automate à entrées et

sorties avec traces complètes optionnelles A représentant le comportement souhaité du système complet et un ensemble In vérifiant $(I_A \setminus I_C) \cup (O_C \setminus O_A) \subseteq In \subseteq I_A \cup O_C$ qui représente l'ensemble des entrées désiré pour la composante à concevoir. L'algorithme se compose de six étapes. Dans la première étape, nous construisons la composition de C, Sol_S et $\widetilde{IOA_A}$ puis on associe à chaque état de la composition les contraintes associées à l'état correspondant dans IOA_A. Dans la deuxième étape, nous traitons les états à partir desquelles une contrainte est non satisfaite ainsi que les transitions non contrôlable qui mènent à l'état *Fail*, c'est-à-dire, les transitions étiquetées par une action qui ne fait pas partie des sorties de la composante à concevoir. Dans la troisième étape, on associe à chaque état contraint c un automate à entrées et sorties dont l'ensemble des traces est égale au sous ensemble de $Tr_{IOA_R}(c)$ contenant les traces sans action externe ainsi que les traces qui contiennent une seul action externe (une sortie) comme dernier élément. Ces automate à entrées et sorties nous permettront dans les étapes ultérieure de caractériser les traces obligatoires ainsi que les traces complètes optionnelles de la solution recherchée. Dans la quatrième étape, nous remplaçons toutes les actions dont les étiquettes ne font pas partie de l'alphabet de la composante à concevoir par l'action interne τ, et nous déterminons l'automate à entrées et sorties déterministe correspondant. Par la suite nous associons à chaque état dans le nouveau automate les contraintes correspondantes. Dans la cinquième étape, nous éliminons de façon récursive toutes les traces non conformes. Finalement, dans la sixième étape nous construisons la solution générique sous la forme d'un automate à entrées et sorties avec traces complètes optionnelles *Sol*.

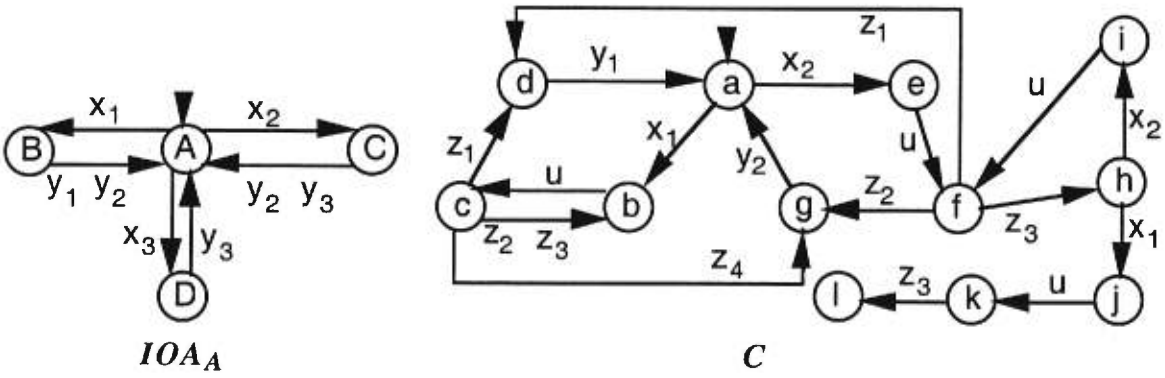


Figure 7.4 : Les automates à entrées et sorties IOA_A et C.

Afin d'illustrer les traitements réalisées dans chaque étape de l'algorithme, nous utiliserons l'exemple suivant. Nous considérons l'automate à entrées et sorties C pour la spécification du contexte avec $I_C=\{x_1, x_2, z_1, z_2, z_3, z_4\}$ et $O_C=\{u, y_1, y_2\}$, l'automate à

entrées et sorties avec traces complètes optionnelles A pour la spécification du système désiré avec $I_{IOA_A}=\{x_1, x_2, x_3\}$, $O_{IOA_A}=\{y_1, y_2, y_3\}$, $MT_A=\{(A, \emptyset), (B, \{\{y_1\}\}), (C, \{\{y_2\}\}), (D, \{\{y_3\}\})\}$, et $OCT_A=\{(A, \emptyset), (B, \{\{y_1, y_2\}\}), (C, \{\{y_2, y_3\}\}), (D, \{\{y_3\}\})\}$, et l'ensemble des entrées $In=\{x_1, x_3, u\}$.

Puisque l'algorithme requiert comme entrée la solution générique pour la relation de conformité réalisation sécuritaire, nous avons utilisé Algorithme 7.1 pour obtenir l'automate à entrées et sorties $Sol_{\S}$ illustré dans Figure 7.5.

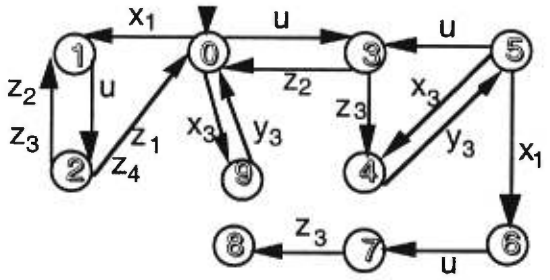


Figure 7.5 : L'automate à entrées et sorties $Sol_{\S}$.

Algorithme 7.2

Entrée : La spécification du contexte C sous forme d'un automate à entrées et sorties, la spécification du système A sous forme d'un automate à entrées et sorties avec traces complètes optionnelles, un ensemble d'entrées pour la solution In, et la spécification de la solution générique pour la relation de conformité réalisation sécuritaire $Sol_{\S}$ sous forme d'un automate à entrées et sorties.

Sortie : Un automate à entrées et sorties avec traces complètes optionnelles Sol avec $I_{IOA_{Sol}}=In$ qui satisfait l'équation $(C||IOA_{Sol})\leq_{conf}A$ si une solution existe.

Étape 1. Dans cette étape, nous construisons un automate à entrées et sorties avec traces complètes optionnelles R tel que IOA_R est la composition de C, $Sol_{\S}$ et $\widetilde{IOA_A}$, puis nous initialisons pour tout état de IOA_R les ensembles $MT_R(c)$ et $OCT_R(c)$ par l'ensemble vide. Si $MT_A(s_{oIOA_A})$ est non vide, nous l'affectons à $MT_R(s_{oIOA_R})$. Pour chaque état $c=(s_1, s_2, s_3)$ dans $S_{IOA_R}\backslash\{s_{oIOA_R}\}$ tel que $entering(c)\cap(I_{IOA_A}\cup O_{IOA_A})$ et $MT_A(s_3)$ sont non vides, nous affectons à $MT_R(c)$ l'ensemble $MT_A(s_3)$. Ceci signifie que pour tout état c de IOA_R ayant un ensemble $MT_R(c)$ non vide une propriété de progrès doit être satisfaite. La complexité dans le pire cas de cette étape est polynomiale en fonction du nombre d'états de IOA_A et C, et du nombre d'éléments dans l'alphabet de $IOA_A\,||C$ (c'est-à-dire, $O(nmr^2)$ où n=nombre d'états de A, m=nombre d'états de C et r=nombre d'éléments dans l'alphabet de $IOA_A\,||C$).

IOA_R=C||$Sol_{\mathfrak{S}}$||$\widetilde{IOA_A}$;
POUR chaque état c=(s_1, s_2, s_3) dans S_{IOA_R} **FAIRE** $MT_R(c)$:=Ø; $OCT_R(c)$:=Ø;
SI $MT_A(s_{oIOA_A})$≠Ø **ALORS** $MT_R(s_{oIOA_R})$:=$MT_A(s_{oIOA_A})$;
POUR chaque état c=(s_1, s_2, s_3) dans S_{IOA_R} \{s_{oIOA_R}} **FAIRE**
SI *entering*(c)∩(I_{IOA_A}∪O_{IOA_A})≠Ø et $MT_A(s_3)$≠Ø **ALORS** $MT_R(c)$:=$MT_A(s_3)$;

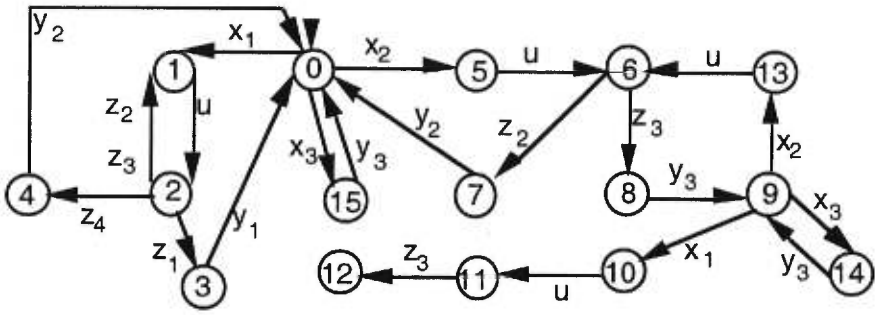


Figure 7.6 : L'automate à entrées et sorties IOA_R obtenu à la fin de l'étape 1

Pour notre exemple, nous obtenons à la fin de l'étape 1 l'automate à entrées et sorties IOA_R illustré dans Figure 7.6 et les ensembles de contraintes MT_R={(0, Ø), (1, {{y_1}}), (2, Ø), (3, Ø), (4, Ø), (5, {{y_2}}), (6, Ø), (7, Ø), (8, Ø), (9, Ø), (10, {{y_1}}), (11, Ø), (12, Ø), (13, {{y_2}}), (14, {{y_3}}), (15, {{y_3}})}.

Étape 2. Un état accessible à partir d'un état c de IOA_R grâce à une trace composée seulement par des actions internes (non observables par l'environnement) est appelé successeur interne de c; de plus l'ensemble contenant toutes les sorties externes présentes dans les successeurs internes de c est noté ext-out-after(c); l'état c est dit silencieux si ext-out-after(c) est vide.

Dans cette étape, nous éliminons de IOA_R certaines traces non conformes. Pour tout état c de IOA_R où $MT_R(c)$ est non vide, nous vérifions d'abord si les contraintes imposées par $MT_R(c)$ sont satisfaites. Par la suite nous éliminons les successeurs internes silencieux de c. Enfin, nous éliminons toutes les transitions étiquetées par une action non contrôlable. À la fin de cette étape, tous les états contraints satisfont leurs contraintes et ne peuvent avoir que l'état $Fail_R$ comme successeur interne silencieux; de plus, toutes les transitions menant à l'état $Fail_R$ sont étiquetées par des actions contrôlables, c'est-à-dire, des sorties de la composante à concevoir.

Le déroulement de cette étape est comme suit. Nous construisons deux ensembles : l'ensemble *NonSilentStates* contenant les états de IOA_R où au moins une sortie externe est présente; et l'ensemble *ConstrainedStates* qui contient au départ tous les états c de IOA_R pour

lesquelles $MT_R(c)$ n'est pas vide. Nous répétons le traitement suivant tant que l'ensemble *ConstrainedStates* n'est pas vide. Nous enlevons de l'ensemble *ConstrainedStates* un élément c, puis nous déterminons l'ensemble de ses successeurs internes, noté *Succint*(c), ainsi que l'ensemble ext-out-after(c); si il existe un élément dans $MT_R(c)$ tel que son intersection avec ext-out-after(c) est vide, alors nous retournons "PAS DE SOLUTION" dans le cas où c est l'état initial, sinon nous remplaçons chaque transition $s\text{-}\ t \rightarrow c$ étiquetée par une action externe t par $s\text{-}\ t \rightarrow Fail_R$, nous affectons à $MT_R(c)$ l'ensemble vide et nous remplaçons IOA_R par sa composante connexe contenant l'état initial. Maintenant, si toutes les contraintes imposées par $MT_R(c)$ ont été satisfaite, c'est-à-dire, $MT_R(c)$ n'est pas vide, nous nous intéressons aux états silencieux dans *Succint*(c); pour chaque état c' de ce genre, nous retournons "PAS DE SOLUTION" dans le cas où c' est l'état initial, sinon chaque transition $s\text{-}\ t \rightarrow c'$ est remplacée par $s\text{-}\ t \rightarrow Fail_R$ puis nous remplaçons IOA_R par sa composante connexe contenant l'état initial. Par la suite, pour chaque transition $s\text{-}\ t \rightarrow Fail_R$ avec t dans $(I_{IOA_A} \cup O_C)$, nous retournons "PAS DE SOLUTION" si s est l'état initial, sinon chaque transition $s'\text{-}\ t \rightarrow s$ est remplacée par $s'\text{-}\ t \rightarrow Fail_R$ puis nous remplaçons IOA_R par sa composante connexe contenant l'état initial. Enfin, nous remettons à jour l'ensemble *ConstrainedStates* en lui affectant l'ensemble contenant les états c de IOA_R pour lesquels $MT_R(c)$ n'est pas vide.

La complexité dans le pire cas de cette étape est polynomiale en fonction du nombre d'états de IOA_A et C, et du nombre d'éléments dans l'alphabet de $IOA_A \,\|C$ (c'est-à-dire, $O(n^3m^3r^3)$ où n=nombre d'états de A, m=nombre d'états de C et r=nombre d'éléments dans l'alphabet de $IOA_A \,\|C$).

Procédure *ÉliminerCertainesTracesNonConformes*

IntActions :=$(I_C \cup O_C) \setminus (I_{IOA_A} \cup O_{IOA_A})$;
NonSilentStates=$\{s \in S_{IOA_R} \mid out(s) \cap O_{IOA_A} \neq \emptyset\}$;
ConstrainedStates := $\{c \in S_{IOA_R} \mid MT_R(c) \neq \emptyset\}$;
TANT QUE *ConstrainedStates* $\neq\emptyset$ **FAIRE**
  c := un élément de *ConstrainedStates* ;
  ConstrainedStates := *ConstrainedStates* $\setminus\{c\}$;
  Succint(c)=$\{c' \in S_R \setminus \{Fail_R\} \mid \exists \sigma \in$ IntActions* telle que $c'=c_\sigma\}$;
  ext-out-after(c) := $\cup_{c' \in Succint(c)}\ out(c') \cap O_{IOA_A}$;
  TempMT :=$MT_R(c)$;
  TANT QUE TempMT$\neq\emptyset$ **FAIRE**
    Y := un élément de TempMT;
    TempMT := TempMT$\setminus\{Y\}$;

SI $Y \cap ext\text{-}out\text{-}after(c)=\emptyset$ **ALORS** // une contrainte pour c n'est pas satisfaite
SI $c=s_{0IOA_R}$ **ALORS** retourner "PAS DE SOLUTION"; **STOP**;
SINON
remplacer $s\text{-}\ t \rightarrow c$ par $s\text{-}\ t \rightarrow Fail_R$ pour tout t dans
$entering(c) \cap (I_{IOA_A} \cup O_{IOA_A})$;
$MT_R(c):=\emptyset$; TempMT:=$\emptyset$; $IOA_R:=CC(IOA_R)$;
FinTantQue TempMT$\neq\emptyset$
SI $MT_R(c)\neq\emptyset$ **ALORS**
TANT QUE $Succint(c)\neq\emptyset$ **FAIRE**
c' := un élément de $Succint(c)$;
$Succint(c) := Succint(c)\backslash\{c'\}$;
SI $Succint(c') \cap NonSilentStates=\emptyset$ **ALORS**
SI $c'=s_{0IOA_R}$ **ALORS** retourner "PAS DE SOLUTION"; **STOP**;
SINON
remplacer $s\text{-}\ t \rightarrow c'$ par $s\text{-}\ t \rightarrow Fail_R$ pout tout t dans $entering(c')$;
$IOA_R:=CC(IOA_R)$; $Succint(c) := Succint(c) \cap S_{IOA_R}$;
FinTantQue $Succint(c)\neq\emptyset$
SI $entering(Fail_R) \cap (I_{IOA_A} \cup O_C)\neq\emptyset$ **ALORS**
TempTrans :=$\{(s, t, Fail_R) \in T_{IOA_R} \mid t \in (I_{IOA_A} \cup O_C)\}$;
TANT QUE TempTrans$\neq\emptyset$ **FAIRE**
$(s', t', Fail_R)$:= un élément de TempTrans;
SI $s'=s_{0IOA_R}$ **ALORS** retourner "PAS DE SOLUTION"; **STOP**;
SINON
remplacer $s''\ -\ t'' \rightarrow s'$ par $s''\ -\ t''\ \rightarrow Fail_R$ pour tout t'' dans $entering(s')$;
$IOA_R:=CC(IOA_R)$;
TempTrans :=$\{(s, t, Fail_R) \in T_{IOA_R} \mid t \in (I_{IOA_A} \cup O_C)\}$;
FinTantQue TempTrans$\neq\emptyset$
$ConstrainedStates := \{c \in S_{IOA_R} \mid MT_R(c)\neq\emptyset\}$;
FinTantQue $ConstrainedStates \neq\emptyset$

Pour notre exemple, les traitements réalisés dans l'étape 2 sont les suivants. À partir de l'état 10, la sortie externe y_2 requise par les contraintes n'est pas possible; cet état est donc remplacé par $Fail_R$. Par la suite, comme dans l'état 9 une transition ($9\text{-}x_1\text{->}Fail_R$) étiquetée par une action non contrôlable mène à l'état $Fail_R$, l'état 9 est lui même remplacé par $Fail_R$. L'automate obtenu à la fin de cette étape est illustré dans Figure 7.7, et les contraintes

associées à ses états sont MT_R={(0, Ø), (1, {{y_1}}), (2, Ø), (3, Ø), (4, Ø), (5, {{y_2}}), (6, Ø), (7, Ø), (8, Ø), (15, {{y_3}})}.

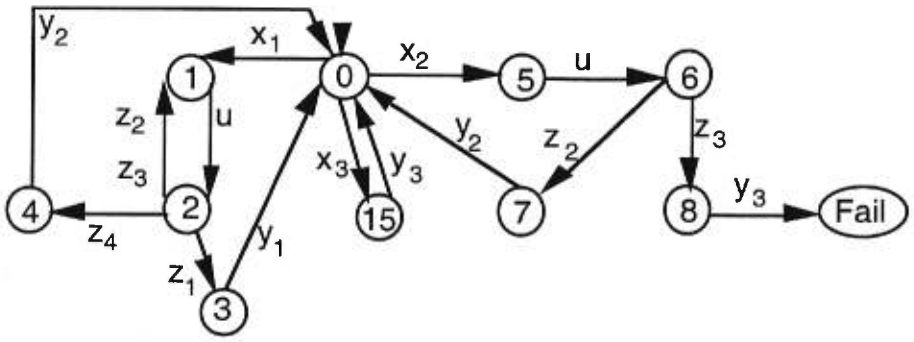


Figure 7.7 : L'automate à entrées et sorties IOA_R obtenu à la fin de l'étape 2

Étape 3. Dans cette étape, on associe à chaque état contraint c un automate à entrées et sorties $CONST(c)$ dont l'ensemble des traces est égale au sous ensemble de $Tr_{IOA_R}(c)$ contenant les traces sans action externe ainsi que les traces qui contiennent une seul action externe (une sortie) comme dernier élément. Ces automate à entrées et sorties nous permettront dans les étapes ultérieure de caractériser les traces obligatoires ainsi que les traces complètes optionnelles de la solution recherchée. Afin de pouvoir composer $CONST(c)$ avec l'automate à entrées et sorties obtenu à partir de IOA_R après avoir caché certaines actions et aussi préserver toutes les sorties externes possibles après c dans $CONST(c)$, nous affectons l'ensemble vide à $O_{CONST(c)}$ et l'ensemble contenant toutes les actions présentes dans IOA_R à $I_{CONST(c)}$.

La complexité dans le pire cas de cette étape est polynomiale en fonction du nombre d'états de IOA_R (c'est-à-dire, $O(n^2m^2r)$ où n=nombre d'états de A, m=nombre d'états de C et r=nombre d'éléments dans l'alphabet de $IOA_A \| C$).

Procédure *SauvegardeDesContraintes*

POUR chaque état c dans $S_{IOA_R}\backslash\{Fail_R\}$ tel que $MT_R(c)\neq\emptyset$ **FAIRE**

IntActions :=$(I_C\cup O_C)\backslash(I_{IOA_A}\cup O_{IOA_A})$;

TempStates := $\{c'\in S_{IOA_R} \mid \exists\sigma\in$ IntActions* telle que $c'=c_\sigma\}$;

TempTrans := $\{(s, t, s')\in Tr_{IOA_R} \mid s, s'\in$ TempStates et $t\in$ IntActions$\}$;

TempTransFinal := $\{(s, t, Final) \mid s\in$ TempStates, $t\in leaving(s)\cap O_{IOA_A}$ et $s_t\neq Fail_R\}$;

TempTransFail := $\{(s, t, Fail_R)\in Tr_{IOA_R} \mid s\in$ TempStates et $t\in O_{IOA_A}\}$;

SI TempTransFinal≠Ø **ALORS** TempStates := TempStates$\cup\{Final\}$;

SI TempTransFail≠Ø **ALORS** TempStates := TempStates$\cup\{Fail_R\}$;

TempTrans := TempTrans∪TempTransFinal∪TempTransFail;
$CONST(c)$:=(TempStates, $I_{IOA_R} \cup O_{IOA_R}$, Ø, TempTrans, c);

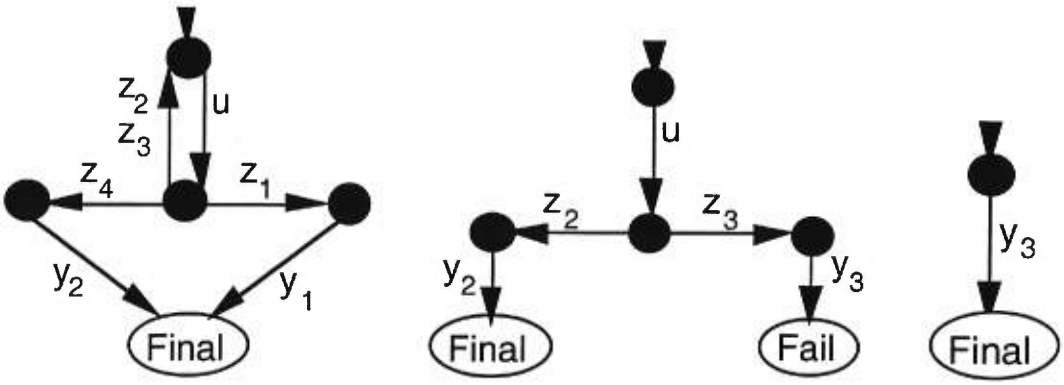


Figure 7.8 : Les automates à entrées et sorties $CONST(1)$, $CONST(5)$ et $CONST(15)$.

Étape 4. Dans cette étape, nous remplaçons dans IOA_R toutes les actions dont les étiquettes ne font pas partie de l'alphabet de la composante à concevoir par l'action interne τ, ensuite nous déterminons l'automate à entrées et sorties déterministe IOA_{R_1} tel que $Tr_{IOA_{R_1}}=Tr_{IOA_R}$. Puisque un état de IOA_{R_1} correspond à un sous ensemble d'états de IOA_R, nous remplaçons tout état de IOA_{R_1} qui contient $Fail_R$ par $Fail_{R_1}$. De plus, nous affectons à l'ensemble TempConstraint($s_{oIOA_{R_1}}$) l'ensemble des couples ($MT_R(c)$, $CONST(c)$) pour tout état c dans $s_{oIOA_{R_1}}$ tel que $MT_R(c)$ n'est pas vide. Pour tout état s dans $S_{IOA_{R_1}} \setminus \{s_{oIOA_{R_1}}, Fail_{R_1}\}$, si $entering(s)$ contient des actions externes, nous affectons à l'ensemble TempConstraint(s) l'ensemble des couples ($MT_R(c)$, $CONST(c)$) pour tout état c dans s tel que $MT_R(c)$ n'est pas vide, sinon nous affectons à l'ensemble TempConstraint(s) l'ensemble des couples ($MT_R(c)$, $CONST(c)$) pour tout état c dans s tel que $MT_R(c)$ n'est pas vide et c est accessible dans IOA_R à partir d'un état c'' dans s avec une action externe dans $(I_{IOA_A} \cup O_C) \setminus I_n$, c'est-à-dire, ils existent s' dans R_1, c' dans s' et c'' dans s tels que $s=s'_t$, $c''=c'_{t.\sigma}$ dans IOA_R et $c=c''_u$ pour σ dans $((I_{IOA_A} \cup O_C) \setminus I_n)^*$ et u dans $(I_{IOA_A} \cup O_C) \setminus I_n$.

La complexité dans le pire cas de cette étape est exponentielle en fonction du nombre des états de IOA_R. On note par ε le mot vide.

Procédure *ÉliminerActionsNonVisibleParComposante*

ConstrainedStates := $\{c \in S_{IOA_R} \mid MT_R(c) \neq \emptyset\}$;
$s_{oIOA_{R_1}} := \{c \in S_{IOA_R} \mid \exists \sigma \in ((I_{IOA_A} \cup O_C) \setminus In)^*$ telle que $c=(s_{oIOA_R})\sigma\}$;
TempConstraint($s_{oIOA_{R_1}}$) := $\{(MT_R(c), CONST(c)) \mid c \in s_{oIOA_{R_1}} \cap ConstrainedStates\}$;
$S_{IOA_{R_1}} := \{s_{oIOA_{R_1}}\}$; $T_{IOA_{R_1}} := \emptyset$;

TempStates := $\{s_{oIOA_{R_1}}\}$;

L :=$In \cup (I_C \setminus I_{IOA_A}) \cup (O_{IOA_A} \setminus O_C)$;

TANT QUE TempStates≠Ø **FAIRE**

s := un élément de TempStates;

TempStates:=TempStates\{s};

POUR chaque t dans L **FAIRE**

POUR chaque c dans s **FAIRE**

$Succh(c)=\{c' \in S_{IOA_R} \mid \exists \sigma \in ((I_{IOA_A} \cup O_C) \setminus I_n)^+$ telle que $c'=c_{t.\sigma}\}$;

$s' := \cup_{c \in s} Succh(c)$;

$s'' := s' \cup \{c_t \mid c \in s\}$;

SI $s'' \neq \emptyset$ **ALORS**

SI $Fail_R \in s''$ **ALORS** $T_{IOA_{R_1}} := T_{IOA_{R_1}} \cup \{(s, t, Fail_{R_1})\}$;

$S_{IOA_{R_1}} := S_{IOA_{R_1}} \cup \{Fail_{R_1}\}$;

SINON

$T_{IOA_{R_1}} := T_{IOA_{R_1}} \cup \{(s, t, s'')\}$;

SI $s'' \notin S_{IOA_{R_1}}$ **ALORS**

$S_{IOA_{R_1}} := S_{IOA_{R_1}} \cup \{s''\}$; TempStates:=TempStates$\cup\{s''\}$;

TempConstraint(s''):=Ø;

SI $t \in ((O_C \setminus O_{IOA_A}) \cup (I_C \setminus I_{IOA_A}))$ **ALORS**

TempConstraint(s''):=$\{(MT_R(c), CONST(c)) \mid c \in s' \cap ConstrainedStates\} \cup$TempConstraint($s''$);

SINON

TempConstraint(s''):=$\{(MT_R(c), CONST(c)) \mid c \in s'' \cap ConstrainedStates\}$;

FinTantQue TempStates≠Ø

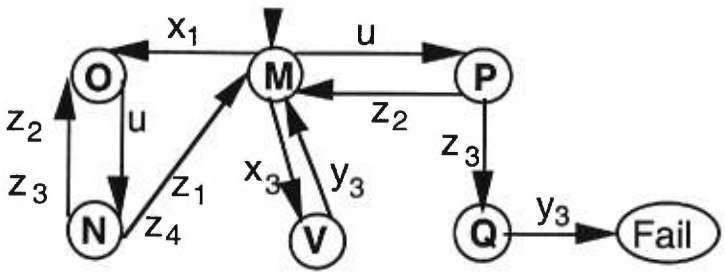


Figure 7.9 : L'automate à entrées et sorties IOA_{R_1}

À la fin de l'étape 4, l'automate à entrées et sorties IOA_{R_1} obtenu dans le cas de notre

exemple est illustré dans la Figure 7.9, de plus nous avons les contraintes suivantes : TempConstraint(**M**)={({$\{y_2\}$}, *CONST*(5))}, TempConstraint(**O**)={({$\{y_1\}$}, *CONST*(1))} et TempConstraint(**V**)={({$\{y_3\}$}, *CONST*(15))}.

Étape 5. Dans cette étape, nous éliminons de façon récursive les traces menant à $Fail_{R_1}$. Pour la réalisation de ce but, nous initialisons tout d'abord l'ensemble *ConstrainedStates* par l'ensemble contenant chaque état c de IOA_{R_1} pour lequel TempConstraint(c) n'est pas vide, et nous éliminons toutes les transitions menant à $Fail_{R_1}$. Par la suite nous répétons le traitement suivant tant que l'ensemble *ConstrainedStates* n'est pas vide. Nous éliminons de *ConstrainedStates* un élément s, et nous affectons TempConstraint(s) à un ensemble temporaire Temp. Tant que Temp n'est pas vide, nous en retirons un élément ($MT_R(c)$, *CONST*(c)); nous affectons à *CONST*(c) sa composition avec l'automate à entrées et sorties obtenu à partir de IOA_{R_1} en considérant s comme l'état initial. Si il existe un élément dans $MT_R(c)$ tel que son intersection avec l'ensemble des sorties externes présentes dans *CONST*(c) est vide, nous retournons "PAS DE SOLUTION" dans le cas où s est l'état initial de IOA_{R_1}, sinon nous remplaçons chaque transition s'-$t \rightarrow s$ par s'-$t \rightarrow Fail_{R_1}$ puis nous remplaçons IOA_{R_1} par sa composante connexe contenant l'état initial et nous affectons à Temp l'ensemble vide. Si l'intersection de *entering*($Fail_{R_1}$) et *In* n'est pas vide, pour chaque transition s''-$t \rightarrow Fail_{R_1}$ avec t dans l'intersection, nous retournons "PAS DE SOLUTION" dans le cas où s'' est l'état initial, sinon nous remplaçons chaque transition s'-$t \rightarrow s''$ par s'-$t \rightarrow Fail_{R_1}$ puis nous remplaçons IOA_{R_1} par sa composante connexe contenant l'état initial. Par la suite nous mettons à jour l'ensemble *ConstrainedStates* en lui affectons l'ensemble contenant les états c de IOA_{R_1} pour lesquels TempConstraint(c) n'est pas vide puis nous éliminons toutes les transitions menant à $Fail_{R_1}$. Si s est encore un élément de $S_{IOA_{R_1}}$, nous nous intéressons aux états dans *CONST*(c), différent de c et *Final*, à partir desquels on ne peut pas atteindre au moins un état où une sorties externe est présente. Pour chaque tel état c' dans $S_{CONST(c)}$, nous remplaçons chaque transition c''- $t \rightarrow c'$ par c''- $t \rightarrow Fail_{CONST(c)}$; puis nous remplaçons *CONST*(c) par sa composante connexe contenant l'état initial. Maintenant pour chaque transition c''- $t \rightarrow Fail_{CONST(c)}$, nous déterminons une trace σ telle que $c_\sigma = c''$ et nous remplaçons la transition s_σ-$t \rightarrow s'$ par s_σ-$t \rightarrow Fail_{R_1}$. Si l'intersection de *entering*($Fail_{R_1}$) et *In* est non vide, pour chaque transition s''-$t \rightarrow Fail_{R_1}$ avec t dans l'intersection, nous retournons "PAS DE SOLUTION" dans le cas où s'' est l'état initial, sinon nous remplaçons chaque transition s'-$t \rightarrow s''$ par s'-$t \rightarrow Fail_{R_1}$ puis nous remplaçons IOA_{R_1} par sa composante connexe contenant l'état initial. Enfin, nous remettons à jour l'ensemble *ConstrainedStates* en lui affectant l'ensemble contenant les états de IOA_{R_1} pour lesquels TempConstraint(c) n'est

pas vide et nous éliminons toutes les transitions menant à $Fail_{R_1}$.

La complexité dans le pire cas de cette étape est polynomiale en fonction du nombre des états de IOA_{R_1} (c'est-à-dire, $O(n^5r^2)$ où n=nombre d'états de IOA_{R_1} et r=nombre d'éléments dans l'alphabet de IOA_{R_1}).

Procédure *ÉliminerÉtatFail*

ConstrainedStates := $\{c \in S_{IOA_{R_1}} \mid \text{TempConstraint}(c) \neq \emptyset\}$;
$I_{IOA_{R_1}} := In$; $O_{IOA_{R_1}} := (I_C \backslash I_{IOA_A}) \cup (O_{IOA_A} \backslash O_C)$;
$T_{IOA_{R_1}} := T_{IOA_{R_1}} \backslash \{(c, t, Fail_{R_1}) \in T_{IOA_{R_1}}\}$;
TANT QUE *ConstrainedStates* $\neq \emptyset$ **FAIRE**
  s := un élément de *ConstrainedStates* ;
  ConstrainedStates := *ConstrainedStates* $\backslash \{s\}$;
  Temp := TempConstraint(s);
  TANT QUE Temp $\neq \emptyset$ **FAIRE**
    $(MT_R(c), CONST(c))$:= un élément de Temp;
    Temp := Temp$\backslash \{(MT_R(c), CONST(c))\}$;
    $CONST(c) := CONST(c) \| (S_{IOA_{R_1}}, I_{IOA_{R_1}}, O_{IOA_{R_1}}, T_{IOA_{R_1}}, s)$;
    Temp1 := $\cup_{c' \in S_{CONST(c)}} out(c') \cap O_{IOA_A}$;
    Temp2 := $MT_R(c)$;
    TANT QUE Temp2$\neq \emptyset$ **FAIRE**
      Y := un élément de Temp2;
      Temp2 := Temp2$\backslash \{Y\}$;
      SI $Y \cap$Temp1$=\emptyset$ **ALORS**
        SI $s = s_{0IOA_{R_1}}$ **ALORS** retourner "PAS DE SOLUTION"; **STOP**;
        SINON
          remplacer s'- $t \rightarrow s$ par s'- $t \rightarrow Fail_{R_1}$ pour chaque t dans $entering(s)$;
          $IOA_{R_1} := CC(IOA_{R_1})$; Temp:=$\emptyset$; Temp2:=$\emptyset$;
          SI $entering(Fail_{R_1}) \cap In \neq \emptyset$ **ALORS**
            Temp3 := $\{s'' \mid (s'', t, Fail_{R_1}) \in T_{OA_{R_1}}$ et $t \in In\}$;
            TANT QUE Temp3$\neq \emptyset$ **FAIRE**
              s'' := un élément de Temp3;
              SI $s'' = s_{0IOA_{R_1}}$ **ALORS**
                retourner "PAS DE SOLUTION"; **STOP**;
              SINON
                remplacer s' - $t' \rightarrow s''$ par s' - $t' \rightarrow Fail_{R_1}$ pour chaque t' dans

entering(*s''*);

IOA_{R_1}:=*CC*(IOA_{R_1});

Temp3 :={*s''* | (*s''*, *t*, $Fail_{R_1}$)∈ $T_{OA_{R_1}}$ et *t* ∈ *In*};

FinTantQue Temp3≠Ø

ConstrainedStates := {*p*∈ $S_{OA_{R_1}}$ | TempConstraint(*p*)≠Ø};

$T_{IOA_{R_1}}$:= $T_{IOA_{R_1}}$\{(*p*, *t*, $Fail_{R_1}$)∈ $T_{IOA_{R_1}}$};

FinTantQue Temp2≠Ø

SI *s*∈ $S_{IOA_{R_1}}$ **ALORS**

NonSilentStates={*s'*∈ $S_{CONST(c)}$ | *out*(*s'*)∩O_{IOA_A}≠Ø};

Temp4=$S_{CONST(c)}$\{$s_{oCONST(c)}$, *Final*};

TANT QUE Temp4≠Ø **FAIRE**

c' **:=** un élément de Temp4;

Temp4 := Temp4\{*c'*};

SI*Succint*(*c'*)∩*NonSilentStates*=Ø **ALORS**

$S_{CONST(c)}$:= $S_{CONST(c)}$∪{$Fail_{CONST(c)}$};

remplacer *c''*- *t* → *c'* par *c''*- *t* → $Fail_{CONST(c)}$ pour chaque *t* dans *entering*(*c'*);

CONST(*c*):=*CC*(*CONST*(*c*)); Temp4:=Temp4∩$S_{CONST(c)}$;

FinTantQue Temp4≠Ø

SI $Fail_{CONST(c)}$∈ $S_{CONST(c)}$ **ALORS**

Temp5=Ø;

POUR chaque transition *c'*- *t* → $Fail_{CONST(c)}$ **FAIRE**

σ_t := une plus courte trace dans $Tr_{CONST(c)}$ telle que c_{σ_t}=*c'*;

Temp5 := Temp5∪{ σ_t };

TANT QUE Temp5≠Ø **FAIRE**

σ_t **:=** un élément de Temp5;

Temp5 := Temp5\{ σ_t };

remplacer s_{σ_t} - *t* → *s'* par s_{σ_t} - *t* → $Fail_{R_1}$;

FinTantQue Temp5≠Ø

SI *entering*($Fail_{R_1}$)∩*In* ≠Ø **ALORS**

Temp6 :={*s''* | (*s''*, *t*, $Fail_{R_1}$)∈ $T_{OA_{R_1}}$ et *t* ∈ *In*};

TANT QUE Temp6≠Ø **FAIRE**

s'' **:=** un élément de Temp6;

SI *s''*=$s_{oIOA_{R_1}}$ **alors** retourner "PAS DE SOLUTION"; **STOP**;

SINON

remplacer s' - t'→ s'' par s' - t'→ $Fail_{R_1}$ pour chaque t' dans *entering*(s'');

IOA_{R_1}:=$CC(IOA_{R_1})$;

Temp6 :={s'' | (s'', t, $Fail_{R_1}$)∈$T_{OA_{R_1}}$ et $t \in In$};

FinTantQue Temp6≠Ø

Temp:=Ø;

ConstrainedStates := {$p \in S_{IOA_{R_1}}$ | TempConstraint(p)≠Ø};

$T_{IOA_{R_1}} := T_{IOA_{R_1}} \setminus \{(p, t, Fail_{R_1}) \in T_{R_1}\}$;

FinTantQue Temp≠Ø

FinTantQue *ConstrainedStates* ≠Ø

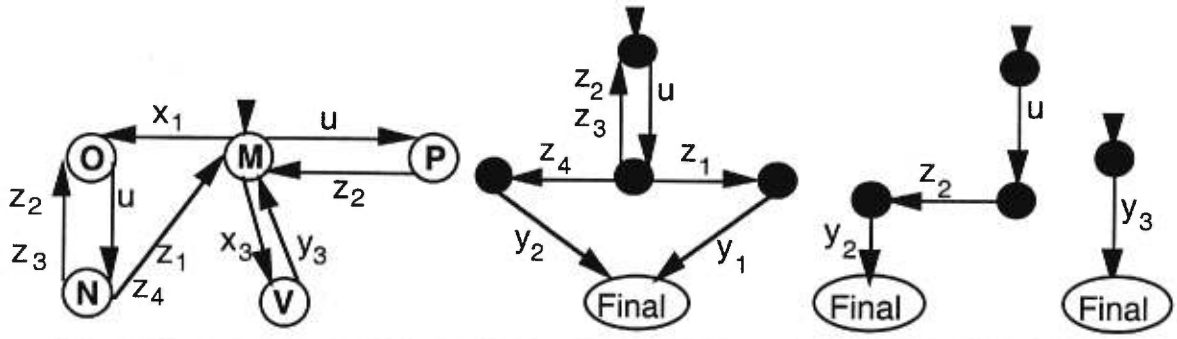


Figure 7.10 : Les automates à entrées et sorties IOA_{R_1}, *CONST*(1), *CONST*(5) et *CONST*(15) à la fin de l'étape 5

Pour notre exemple, les transformations effectuées dans l'étape 5 sont les suivantes. Tout d'abord la transition **Q**-y_3->$Fail_{R_1}$ est éliminée de IOA_{R_1}. Après la mise à jour de l'automate à entrées et sorties *CONST*(5), la transition étiquetée par z_3 mène à un état à partir duquel aucune sortie externe n'est possible. La trace uz_3 est donc utilisée pour remplacer l'état **Q** de IOA_{R_1} par $Fail_{R_1}$. Enfin, après l'élimination de la transition **P**-z_3->$Fail_{R_1}$, toutes les contraintes deviennent satisfaites et les automates à entrées et sorties IOA_{R_1}, *CONST*(1), *CONST*(5) et *CONST*(15) obtenus sont illustrés dans la Figure 7.10.

Étape 6. Dans cette étape, nous construisons la solution générique sous la forme d'un automate à entrées et sorties avec traces complètes optionnelles *Sol* . L'automate à entrées et sorties IOA_{Sol} est égale à IOA_{R_1}. Pour obtenir les ensembles MT_{Sol} et OCT_{Sol} , nous initialisons les ensembles $MT_{Sol}(s)$ et $OCT_{Sol}(s)$ avec l'ensemble vide pour tout état s dans $S_{IOA_{Sol}}$. Si l'ensemble TempConstraint(s) n'est pas vide alors pour chaque élément ($MT_R(c)$, $CONST(c)$), nous affectons à un ensemble temporaire Temp2 l'ensemble des traces menant à l'état *Final* dans $CONST(c)$. Par la suite pour tout élément Y dans $MT_R(c)$, si son

intersection avec l'ensemble des sorties externes présentes dans l'état c est vide nous affectons à $MT_{Sol}(s)$ sa réunion avec la projection sur l'alphabet de IOA_{Sol} du sous ensemble de Temp2 contenant les traces qui se terminent par un élément dans Y. Nous affectons à un ensemble temporaire Temp3 la projection sur l'alphabet de IOA_{Sol} de Temp2, puis nous éliminons de Temp3 le mot vide et les éléments qui n'appartiennent pas à $OCT_{Sol}(s)$ et qui sont des préfixes d'éléments de $OCT_{Sol}(s)$, et de $OCT_{Sol}(s)$ nous éliminons les éléments qui n'appartiennent pas à Temp3 et qui sont des préfixes d'éléments de Temp3. Enfin, nous affectons à $OCT_{Sol}(s)$ sa réunion avec Temp3. À cause de cycles étiquettes seulement par des actions internes, les ensembles $OCT_{Sol}(c)$ et $MT_{Sol}(c)$ peuvent être infinis. Afin de décrire de tels ensembles infinis nous les représentons par des ensembles finis d'expressions régulières ou comme des ensembles finis d'automates à entrées et sorties.

La complexité dans le pire cas de cette étape est polynomiale en fonction du nombre des états de IOA_{R_1}.

Procédure ***ConstructionSolutionGénériqueSol***

$IOA_{Sol} := IOA_{R_1}$;

$OCT_{Sol} := \emptyset$;

$MT_{Sol} := \emptyset$;

$L := In \cup (I_C \setminus I_{IOA_A}) \cup (O_{IOA_A} \setminus O_C)$;

POUR chaque état s dans $S_{IOA_{Sol}}$ **FAIRE**

$OCT_{Sol}(s) := \emptyset$;

$MT_{Sol}(s) := \emptyset$;

TANT QUE TempConstraint(s)$\neq\emptyset$ **FAIRE**

($MT_R(c)$, $CONST(c)$) := un élément de TempConstraint(s);

TempConstraint(s) :=TempConstraint(s)\{($MT_R(c)$, $CONST(c)$)};

Temp1 := $leaving(s_{o CONST(c)}) \cap O_{IOA_A}$;

Temp2 :=$\{\sigma \in Tr_{CONST(c)}(s_{o CONST(c)}) \mid (s_{o CONST(c)})\sigma = Final\}$;

TANT QUE $MT_R(c)\neq\emptyset$ **FAIRE**

Y := un élément dans $MT_R(c)$;

$MT_R(c) := MT_R(c) \setminus \{Y\}$;

SI $Y \cap$Temp1$=\emptyset$ **ALORS**

$MT_{Sol}(s) := MT_{Sol}(s) \cup (Pr_L(\{\sigma \in$ Temp2 $\mid Pr_L\{\sigma) \in Y\})$;

FinTantQue $MT_R(c)\neq\emptyset$

Temp3 :=Pr_L(Temp2)\{ε};

$OCT_{Sol}(s) := OCT_{Sol}(s) \setminus \{\sigma \in OCT_{Sol}(s)$ et $\sigma \notin$ Temp3 et σ est un préfixe propre d'un élément dans Temp3};

Temp3 :=Temp3\{ $\sigma \in$ Temp3 et $\sigma \notin OCT_{Sol}(s)$ et σ est un préfixe propre d'un élément dans $OCT_{Sol}(s)$};

$OCT_{Sol}(s) := OCT_{Sol}(s) \cup$ Temp3;

FinTantQue TempConstraint(s)$\neq\emptyset$

$MT_{Sol}(s) := \{Y \cap OCT_{Sol}(s) \mid Y \in MT_{Sol}(s)\}$;

$MT_{Sol}(s) := MT_{Sol}(s) \setminus \{Y \in MT_{Sol}(s) \mid$ il existe $Y' \in MT_{Sol}(s)$ tel que $Y' \subset Y\}$;

$OCT_{Sol}(s) := OCT_{Sol}(s) \setminus \{\sigma \in OCT_{Sol}(s) \mid$ longueur de σ est égale à 1$\}$;

$OCT_{Sol} := OCT_{Sol} \cup \{(s, OCT_{Sol}(s))\}$;

$MT_{Sol} := MT_{Sol} \cup \{(s, MT_{Sol}(s))\}$;

FinPour

$Sol := (IOA_{Sol}, MT_{Sol}, OCT_{Sol})$;

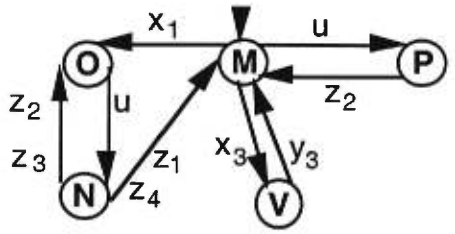


Figure 7.11 : L'automate à entrées et sorties IOA_{Sol}.

Pour notre exemple, la solution générique *Sol* est obtenue sous la forme d'un automate à entrées et sorties avec traces complètes optionnelles pour lequel IOA_{Sol} est illustré dans Figure 7.11, et les ensembles de contraintes sont :

$MT_{Sol} = \{(\mathbf{M}, \{\{uz_2\}\}), (\mathbf{O}, \{\{u(z_2u + z_3u)^*z_1\}\}), (\mathbf{N}, \emptyset), (\mathbf{P}, \emptyset), (\mathbf{V}, \{\{y_3\}\})\}$, et

$OCT_{Sol} = \{(\mathbf{M}, \{uz_2\}), (\mathbf{O}, \{u(z_2u + z_3u)^*z_1, u(z_2u + z_3u)^*z_4\}), (\mathbf{N}, \emptyset), (\mathbf{P}, \emptyset), (\mathbf{V}, \emptyset)\}$.

Théorème 7.3 : Étant donnés un automate à entrées et sorties déterministe C et un automate à entrées et sorties avec traces complètes optionnelles déterministe A, et étant donné un ensemble In tel que $(I_A \setminus I_C) \cup (O_C \setminus O_A) \subseteq In \subseteq I_A \cup O_C$, Si l'algorithme 7.2 produit un automate à entrées et sorties avec traces complètes optionnelles *Sol* alors $(C \| IOA_{Sol}) \leq_{conf} A$ avec $I_{Sol_S} = In$, sinon il n'existe aucune solution pour l'équation $(C \| X) \leq_{conf} A$ ayant In comme ensemble d'entrées.

Preuve du Théorème 7.3 :

Première partie : Si Algorithme 7.2 produit un automate *Sol* alors $(C \| IOA_{Sol}) \leq_{conf} A$

Si l'algorithme 7.2 produit un automate à entrées et sorties avec traces complètes optionnelles *Sol* alors nous devons prouver que $(C \| IOA_{Sol}) \leq_{conf} A$. Par définition de la

relation $\leq_{conf}$, ceci est équivalent à $(C\|IOA_{Sol})\leq_{\$} IOA_A$ et $(C\|IOA_{Sol})\leq_{\mathbb{P}} A$.

(a) Preuve de $(C\|IOA_{Sol})\leq_{\$} IOA_A$

$(C\|IOA_{Sol})\leq_{\$} IOA_A$ est équivalent à $IOA_{Sol}\leq_{\$} Sol_{\$}$ d'après théorème 7.2.

Considérons $\sigma\in(I_{Sol_{\$}}\cup O_{Sol_{\$}})^*$, par construction IOA_{Sol} est trace inclus dans $Sol_{\$}$

Donc $\sigma\in Tr_{Sol_{\$}}.O_{Sol_{\$}} \wedge \sigma\in Tr_{IOA_{Sol}} \Rightarrow \sigma\in Tr_{Sol_{\$}}$

Posons $\sigma=\sigma_1.t$, avec $t\in I_{Sol_{\$}}$ et supposons que

$$\sigma\in Tr_{Sol_{\$}} \wedge \sigma\in Tr_{IOA_{Sol}}.I_{Sol_{\$}} \wedge \sigma\notin Tr_{IOA_{Sol}}$$

Donc t a été éliminé de $Tr_{(C\|Sol_{\$}\|\widetilde{IOA_A})}$, mais $t\in I_{Sol_{\$}}$ implique $\sigma_1\notin Tr_{IOA_{Sol}}$, contradiction.

Donc $IOA_{Sol}\leq_{\$} Sol_{\$}$

(b) Preuve de $(C\|IOA_{Sol})\leq_{\mathbb{P}} A$

Pour la preuve de $(C\|IOA_{Sol})\leq_{\mathbb{P}} A$, nous devons vérifier les points (i), (ii), (iii) et (iv) de Définition 3.12.

Si $MT_A(s_{0IOA_A})\neq\emptyset$ alors vérifions les points (i) et (ii) de Définition 3.12

(i) supposons qu'il existe $Y\in MT_A((s_{0IOA_A}))$ tel que

$$Pr_{IOA_A}(Tr_{(C\|IOA_{Sol})}((s_{0(C\|IOA_{Sol})})))\cap Y=\emptyset$$

Donc dans l'étape 2 ou l'étape 5 de Algorithme 7.2, "PAS DE SOLUTION" est retournée, contradiction.

(ii) supposons qu'il existe $\sigma_1\in Tr_{(C\|IOA_{Sol})}((s_{0(C\|IOA_{Sol})}))$ telle que

$Pr_{IOA_A}(\sigma_1)=\varepsilon$ et $Pr_{IOA_A}(Tr_{(C\|IOA_{Sol})}((s_{0(C\|IOA_{Sol})})\sigma_1))\cap out((s_{0IOA_A}))=\emptyset$

Donc dans l'étape 2 ou l'étape 5 de Algorithme 7.2, σ_1 sera éliminée de $Tr_{(C\|Sol_{\$}\|\widetilde{IOA_A})}$,

Donc $\sigma_1\notin Tr_{(C\|IOA_{Sol}\|\widetilde{IOA_A})}$ ce qui implique $\sigma_1\notin Tr_{(C\|IOA_{Sol})}$, contradiction.

Vérification des points (iii) et (iv) de Définition 3.12

Considérons $\sigma=\sigma_1.t\in Tr_{(C\|IOA_{Sol})}$ telle que

$t\in(I_{IOA_A}\cup O_{IOA_A})$, $\sigma'=Pr_{IOA_A}(\sigma)\in Tr_{IOA_A}$ et $MT_A((s_{0IOA_A})\sigma')\neq\emptyset$

nous avons $\sigma\in Tr_{(C\|IOA_{Sol}\|\widetilde{IOA_A})}$,

(iii) supposons qu'il existe $Y\in MT_A((s_{0IOA_A})\sigma')$ tel que

$$Pr_{IOA_A}(Tr_{(C\|IOA_{Sol})}((s_{0(C\|IOA_{Sol})})\sigma))\cap Y=\emptyset$$

Donc dans l'étape 2 ou l'étape 5 de Algorithme 7.2, σ mènera à $Fail_{R_1}$ et donc t sera éliminée.

Donc $\sigma\notin Tr_{(C\|IOA_{Sol}\|\widetilde{IOA_A})}$, contradiction.

(iv) supposons qu'il existe $\sigma_2\in Tr_{(C\|IOA_{Sol})}((s_{0(C\|IOA_{Sol})})\sigma)$ telle que

$Pr_{IOA_A}(\sigma_2)=\varepsilon$ et $Pr_{IOA_A}(Tr_{(C\|IOA_{Sol})}((s_{0(C\|IOA_{Sol})})\sigma\sigma_2))\cap out((s_{0IOA_A})\sigma')=\emptyset$

Donc dans l'étape 2 ou l'étape 5 de Algorithme 7.2, $\sigma\sigma_2$ sera éliminée de

$Tr_{(C\|Sol_{\mathbb{S}}\|\widetilde{IOA_A})}$,

Donc $\sigma\sigma_2 \notin Tr_{(C\|IOA_{Sol}\|\widetilde{IOA_A})}$ ce qui implique $\sigma\sigma_2 \notin Tr_{(C\|IOA_{Sol})}$, contradiction.

Donc $(C\|IOA_{Sol}) \leq_{\mathbb{P}} A$.

Nous concluons que $(C\|IOA_{Sol}) \leq_{conf} A$.

Deuxième partie : Si Algorithme 7.2 ne produit pas de solution alors il n'y a aucune solution

Si la solution générique $Sol_{\mathbb{S}}$ n'existe pas, nous avons montré dans Théorème 7.1 qu'il n'existe aucune solution pour l'équation $(C\|X) \leq_{\mathbb{S}} IOA_A$ ayant *In* comme ensemble d'entrées et donc il n'existe aucune solution pour l'équation $(C\|X) \leq_{conf} A$ ayant *In* comme ensemble d'entrées.

Maintenant, supposons que la solution générique $Sol_{\mathbb{S}}$ existe, et que l'algorithme 7.2 retourne "PAS DE SOLUTION", et qu'il existe un automate à entrées et sorties *B* ayant *In* comme ensemble d'entrées tel que $(C\|B) \leq_{conf} A$,

Puisque $(C\|Sol_{\mathbb{S}}) \leq_{\mathbb{S}} IOA_A$ et $(C\|B) \leq_{\mathbb{S}} IOA_A$ donc d'après Théorème 7.2 $B \leq_{\mathbb{S}} Sol_{\mathbb{S}}$,

ceci implique que $Tr_{(C\|B\|\widetilde{IOA_A})} \subseteq Tr_{(C\|Sol_{\mathbb{S}}\|\widetilde{IOA_A})}$

Puisque $(C\|B) \leq_{conf} A$, toute trace éliminée de $Tr_{(C\|Sol_{\mathbb{S}}\|\widetilde{IOA_A})}$ dans l'étape 2 ou l'étape 5, ne peut appartenir à $Tr_{(C\|B\|\widetilde{IOA_A})}$,

Algorithme 7.2 retourne "PAS DE SOLUTION" si l'état initial de $C\|Sol_{\mathbb{S}}\|\widetilde{IOA_A}$ doit être éliminé à cause de l'élimination d'une trace dans l'étape 2 ou l'étape 5,

ceci entraîne que l'état initial de $C\|B\|\widetilde{IOA_A}$ ne peut satisfaire une certaine contrainte, contradiction.

Nous concluons qu'il n'existe aucun automate à entrées et sorties *B* ayant *In* comme ensemble d'entrées tel que $(C\|B) \leq_{conf} A$. □

7.3.2 L'ensemble des solutions

La solution obtenue par Algorithme 7.2 est générique. On peut dériver à partir d'elle l'ensemble des solutions pour l'équation $(C \parallel X) \leq_{conf} A$ sous la contrainte $I_X = In$.

Théorème 7.4 : Étant donnés un automate à entrées et sorties déterministe *C* et un automate à entrées et sorties avec traces complètes optionnelles déterministe *A*, et étant donné un ensemble *In* tel que $(I_A \setminus I_C) \cup (O_C \setminus O_A) \subseteq In \subseteq I_A \cup O_C$, si l'algorithme 7.2 produit un automate à entrées et sorties avec traces complètes optionnelles *Sol* alors pour tout automate à entrées et sorties *B* tel que $I_B = In$ et $O_B = O_{Sol_{\mathbb{S}}}$, les propositions suivante sont équivalentes :

i - $(C\|B) \leq_{conf} A$,

ii - $B \leq_{conf} Sol$.

Preuve du Théorème 7.4 :

Première partie : $(C\|B)\leq_{conf}A \Rightarrow B\leq_{conf}Sol$

(1) La preuve de $B \leq_{\$} IOA_{Sol}$:

Considérons $\sigma\in(I_B\cup O_B)^*$ et soit $|\sigma|$=n, nous prouverons que :

$\sigma\in Tr_{IOA_{Sol}}.(O_B\cup\{\varepsilon\})\wedge \sigma\in Tr_B .(I_B\cup\{\varepsilon\}) \Rightarrow \sigma\in Tr_{(B\|IOA_{Sol})}$

Puisque $(C\|B)\leq_{\$}A$, donc d'après Théorème 7.2, $B \leq_{\$} Sol_{\$}$.

Ceci entraîne que $Tr_{(C\|B\|\widetilde{IOA_A})}\subseteq Tr_{(C\|Sol_{\$}\|\widetilde{IOA_A})}$

Cas 1 : $\sigma\in Tr_{IOA_{Sol}} \wedge \sigma\in Tr_B .I_B$

Puisque $Tr_{IOA_{Sol}}\subseteq Tr_{Sol_{\$}}$ donc $\sigma\in Tr_{Sol_{\$}}\wedge \sigma\in Tr_B .I_B \Rightarrow \sigma\in Tr_B$

Donc $\sigma\in Tr_{(B\|IOA_{Sol})}$.

Cas 2 : $\sigma\in Tr_{IOA_{Sol}}.O_B \wedge \sigma\in Tr_B$

Puisque $Tr_{IOA_{Sol}}\subseteq Tr_{Sol_{\$}}$ donc $\sigma\in Tr_{Sol_{\$}}.O_{Sol_{\$}} \wedge \sigma\in Tr_B \Rightarrow \sigma\in Tr_{Sol_{\$}}$

Posons $\sigma=\sigma_1.t$ avec $t\in O_B$,

Supposons que $\sigma\notin Tr_{IOA_{Sol}}$, puisque $\sigma_1\in Tr_{IOA_{Sol}}$ ceci entraîne que σ a été éliminée de $Tr_{Sol_{\$}}$ dans l'étape 2 ou l'étape 5.

- supposons que σ a été éliminée de $Tr_{Sol_{\$}}$ dans l'étape 2 :

  Donc il existe $\sigma_2.t\in Tr_{(C\|Sol_{\$}\|IOA_A)}$ telle que $Pr_{Sol_{\$}}(\sigma_2.t)=\sigma$

  De plus $\sigma_2.t\in Tr_{(C\|B)}$

  Soit $\sigma_3=\sigma_4.t_1=Pr_A(\sigma_2)$ et $\sigma_2=\sigma_5.t_1.\sigma_6$ telle que $Pr_A(\sigma_5.t_1)=\sigma_3$,

  Il existe $Y\in MT_A((s_{o IOA_A})_{\sigma_3})$ tel que

  $$Y\cap Tr_{(C\|Sol_{\$}\|IOA_A)}(s_{o(C\|Sol_{\$}\|IOA_A)})_{\sigma_5.t_1})=\emptyset$$

  Puisque $(C\|B)\leq_P A$, alors il existe $\sigma_7.t_2\in Tr_{(C\|B)}(s_{o(C\|B)})_{\sigma_5.t_1})$ telle que

  $$Pr_A(\sigma_7.t_2)=t_2\in Y$$

  Donc $\sigma_8=(\sigma_5.t_1.\sigma_7.t_2)\in Tr_{(C\|B\|IOA_A)}$

  Puisque $B \leq_{\$} Sol_{\$}$ et $(C\|Sol_{\$})\leq_{\$}A$ alors $\sigma_8\in Tr_{(C\|Sol_{\$}\|IOA_A)}$

  Contradiction avec le faite que σ a été éliminée de $Tr_{Sol_{\$}}$ dans l'étape 2.

- supposons que σ a été éliminée de $Tr_{Sol_{\$}}$ dans l'étape 5 :

  Donc $\sigma_1.t$ mène à $Fail_{R_1}$ pendant une certaine itération de l'étape 5.

  Puisque $Tr_{(C\|B\|\widetilde{IOA_A})}\subseteq Tr_{(C\|Sol_{\$}\|\widetilde{IOA_A})}$, $C\|B$ n'est pas une implantation conforme de A,

  Contradiction.

Donc nous avons $\sigma\in Tr_{IOA_{Sol}}$ ce qui entraîne que $\sigma\in Tr_{(B\|IOA_{Sol})}$.

Nous concluons que $B \leq_{\$} IOA_{Sol}$.

(2) La preuve de $B \leq_P Sol$:

Puisque B et Sol ont les mêmes alphabets d'entrées et de sorties, on peut lors de la

vérification des propriétés de Définition 3.12 regrouper les propriétés (i) et (iii) ainsi que les propriétés (ii) et (iv).

Considérons $\sigma \in Tr_B$ telle que $\sigma \in Tr_{IOA_{Sol}}$ et $MT_{Sol}((s_0 IOA_{Sol})\sigma) \neq \emptyset$

Soit $Y \in MT_{Sol}((s_0 IOA_{Sol})\sigma)$, par construction de *Sol* ils existent :

$\sigma_1 \in Tr_{(C\|IOA_{Sol}\|IOA_A)}$, $\sigma_2 \in Tr_{IOA_A}$, $Y_2 \in MT_A((s_0 IOA_A)\sigma_2)$,

$Y_1 \subseteq Tr_{(C\|IOA_{Sol}\|IOA_A)}((s_{0(C\|IOA_{Sol}\|IOA_A)})\sigma_1)$ et

$Y_3 \subseteq Tr_{(C\|IOA_{Sol}\|IOA_A)}((s_{0(C\|IOA_{Sol}\|IOA_A)})\sigma_1)$

tels que

$\sigma = Pr_B(\sigma_1)$, $\sigma_2 = Pr_{IOA_A}(\sigma_1)$, $Y = Pr_B(Y_1)$, $OCT_{Sol}((s_0 IOA_{Sol})\sigma) = Pr_B(Y_3)$, $Pr_{IOA_A}(Y_1) \subseteq Y_2$

et pour tout $\beta.t \in Tr_{(C\|IOA_{Sol}\|IOA_A)}((s_{0(C\|IOA_{Sol}\|IOA_A)})\sigma_1)$ avec $t \in O_{IOA_A}$,

$Pr_A(\beta.t) = t \in Y_2 \Rightarrow \beta.t \in Y_1$

$Pr_A(\beta.t) = t \in OCT_A((s_0 IOA_A)\sigma_2) \Rightarrow \beta.t \in Y_3$

Vérification des propriétés (i) et (iii) dans Définition 3.12

Puisque $\sigma_1 \in Tr_{(C\|IOA_{Sol}\|IOA_A)}$ et $Pr_B(\sigma_1) \in Tr_B$ donc $\sigma_1 \in Tr_{(C\|B\|IOA_A)}$

Puisque $(C\|B) \leq_P A$ alors il existe $\sigma_3 = \sigma_4.t_1 \in Tr_{(C\|B)}(s_{0(C\|B)})\sigma_1)$ telle que

$Pr_{IOA_A}(\sigma_3) = t_1 \in Y_2$

Puisque $B \leq_{\$} IOA_{Sol}$ et $(C\|IOA_{Sol}) \leq_{\$} IOA_A$ alors $\sigma_1.\sigma_3 \in Tr_{(C\|IOA_{Sol}\|IOA_A)}$

Donc $\sigma_3 \in Y_1$ ce qui entraîne que $Pr_B(\sigma_3) \in Y$

Donc $Tr_B((s_0 B)\sigma) \cap Y \neq \emptyset$.

Vérification des propriétés (i) et (iii) dans Définition 3.12

Soit $\sigma_5 = \sigma_6.t_2 \in Tr_B((s_0 B)\sigma)$ telle que

$\sigma_5 \in Pref(OCT_{Sol}((s_0 IOA_{Sol})\sigma))$ et $\sigma_5 \notin OCT_{Sol}((s_0 IOA_{Sol})\sigma)$

il existe $\sigma_7 = \sigma_1.\sigma_8.t_2 \in Tr_{(C\|IOA_{Sol}\|IOA_A)}$ telle que $Pr_B(\sigma_8.t_2) = \sigma_5$

Puisque $\sigma_7 \in Tr_{(C\|IOA_{Sol}\|IOA_A)}$ et $Pr_B(\sigma_7) \in Tr_B$ alors $\sigma_7 \in Tr_{(C\|B\|IOA_A)}$

Puisque $(C\|B) \leq_P A$ alors il existe $\sigma_9 = \sigma_{10}.t_3 \in Tr_{(C\|B)}(s_{0(C\|B)})\sigma_7)$ telle que

$Pr_{IOA_A}(\sigma_9) = t_3 \in OCT_A((s_0 IOA_A)\sigma_2)$

Puisque $B \leq_{\$} IOA_{Sol}$ et $(C\|IOA_{Sol}) \leq_{\$} IOA_A$ alors $\sigma_7.\sigma_9 \in Tr_{(C\|IOA_{Sol}\|IOA_A)}$

Donc $\sigma_{10} = \sigma_8.t_2.\sigma_9 \in Y_3$ et ceci entraîne que $Pr_B(\sigma_{10}) \in OCT_{Sol}((s_0 IOA_{Sol})\sigma)$.

Nous concluons que $B \leq_P Sol$.

Deuxième partie : $B \leq_{conf} Sol \Rightarrow (C\|B) \leq_{conf} A$

(1) La preuve de $(C\|B) \leq_{\$} IOA_A$:

Nous avons $B \leq_{\$} IOA_{Sol}$ et $IOA_{Sol} \leq_{\$} Sol_{\$}$, montrons que $B \leq_{\$} Sol_{\$}$

Considérons $\sigma \in (I_B \cup O_B)^*$ et soit $|\sigma| = n$, nous montrons que :

$\sigma \in Tr_{Sol_{\$}}.(O_B \cup \{\varepsilon\}) \wedge \sigma \in Tr_B.(I_B \cup \{\varepsilon\}) \Rightarrow \sigma \in Tr_{(B\|Sol_{\$})}$

Si n>1, nous montrons par induction que $\sigma_{[n-1]} \in Tr_{IOA_{Sol}}$

si $\sigma_{[1]} \in O_B$, puisque $B \leq_{\$} IOA_{Sol}$ alors $\sigma_{[1]} \in O_B \wedge \sigma_{[1]} \in Tr_B \Rightarrow \sigma_{[1]} \in Tr_{IOA_{Sol}}$

Si $\sigma_{[1]} \in I_B$, puisque $IOA_{Sol} \leq_{\$} Sol_{\$}$ alors $\sigma_{[1]} \in I_B \wedge \sigma_{[1]} \in Tr_{Sol_{\$}} \Rightarrow \sigma_{[1]} \in Tr_{IOA_{Sol}}$

Supposons que $\sigma_{[k]} \in Tr_{IOA_{Sol}}$ pour $1 \leq k < n-1$, et posons $\sigma_{[k+1]} = \sigma_{[k]}.t$

Si $t \in O_B$, puisque $B \leq_{\$} IOA_{Sol}$ alors $\sigma_{[k+1]} \in Tr_{IOA_{Sol}}.O_B \wedge \sigma_{[k+1]} \in Tr_B \Rightarrow \sigma_{[k+1]} \in Tr_{IOA_{Sol}}$

Si $t \in I_B$, puisque $IOA_{Sol} \leq_{\$} Sol_{\$}$ alors $\sigma_{[k+1]} \in Tr_{IOA_{Sol}}.I_B \wedge \sigma_{[k+1]} \in Tr_{Sol_{\$}} \Rightarrow \sigma_{[k+1]} \in Tr_{IOA_{Sol}}$

D'après le principe d'induction nous avons $\sigma_{[n-1]} \in Tr_{IOA_{Sol}}$

Cas 1 : $\sigma \in Tr_{Sol_{\$}}.O_{Sol_{\$}} \wedge \sigma \in Tr_B$

Puisque $B \leq_{\$} IOA_{Sol}$ alors $\sigma \in Tr_{IOA_{Sol}}.O_B \wedge \sigma \in Tr_B \Rightarrow \sigma \in Tr_{IOA_{Sol}}$

Et puisque $IOA_{Sol} \leq_{\$} Sol_{\$}$ alors $\sigma \in Tr_{Sol_{\$}}.O_B \wedge \sigma \in Tr_{IOA_{Sol}} \Rightarrow \sigma \in Tr_{Sol_{\$}}$

Donc $\sigma \in Tr_{(B \| Sol_{\$})}$

Cas 2 : $\sigma \in Tr_{Sol_{\$}} \wedge \sigma \in Tr_B .I_B$

Puisque $IOA_{Sol} \leq_{\$} Sol_{\$}$ alors $\sigma \in Tr_{Sol_{\$}} \wedge \sigma \in Tr_{IOA_{Sol}} .I_B \Rightarrow \sigma \in Tr_{IOA_{Sol}}$

Et puisque $B \leq_{\$} IOA_{Sol}$ alors $\sigma \in Tr_{IOA_{Sol}} \wedge \sigma \in Tr_B .I_B \Rightarrow \sigma \in Tr_B$

Donc $\sigma \in Tr_{(B \| Sol_{\$})}$

Donc $B \leq_{\$} Sol_{\$}$ qui est équivalente d'après Théorème 7.2 à $(C \| B) \leq_{\$} IOA_A$

(2) <u>La preuve de $(C \| B) \leq_P A$</u> :

Vérification des propriétés (i) dans Définition 3.12

Supposons que $MT_A(s_{0IOA_A}) \neq \emptyset$ et considérons un $Y \in MT_A(s_{0IOA_A})$,

Par construction de Sol il existe $Y_2 \subseteq Tr_{(C \| IOA_{Sol} \| IOA_A)}$ tel que $Y_2 \neq \emptyset$, et pour tout élément $\beta \in Y_2$ $Pr_{IOA_A}(\beta) \in Y$

(a) $Y_2 \cap O_C \neq \emptyset$

Il existe $y \in Y_2 \cap O_C$ tel que $y \in Tr_{(C \| IOA_{Sol} \| IOA_A)}$

Puisque $(C \| B) \leq_{\$} IOA_A$ alors

$Pr_C(y) \in Tr_C \wedge Pr_B(y) \in Tr_B.I_B \wedge Pr_{IOA_A}(y) \in Tr_{IOA_A} \Rightarrow y \in Tr_{(C \| B \| IOA_A)}$

Donc $Pr_{IOA_A}(Tr_{(C \| B)}) \cap Y \neq \emptyset$

(b) $Y_2 \cap O_C = \emptyset$

$Y_3 = Pr_B(Y_2) \in MT_{Sol}(s_{0IOA_{Sol}})$

Puisque $B \leq_P Sol$, il existe $\sigma_5 \in Y_3$ telle que $\sigma_5 \in Tr_B$

Donc il existe $\sigma_6 \in Y_2$ telle que $Pr_B(\sigma_6) = \sigma_5$

Puisque $\sigma_6 \in Tr_{(C \| IOA_{Sol} \| IOA_A)}$ alors $Pr_C(\sigma_6) \in Tr_C$

De plus $Pr_B(\sigma_6) = \sigma_5 \in Tr_B$, donc $\sigma_6 \in Tr_{(C \| B)}$

Donc $Pr_{IOA_A}(Tr_{(C\|B)})\cap Y\neq\emptyset$

Vérification des propriétés (ii) dans Définition 3.12

Supposons qu'il existe $\sigma''\in Tr_{(C\|B)}$ telle que $Pr_{IOA_A}(\sigma'')=\varepsilon$

Soit $|\sigma''|=n$, nous montrons par induction que $\sigma''\in Tr_{IOA_{Sol}}$

Si $\sigma''_{[1]}\in I_B$, puisque $(C\|IOA_{Sol})\leq_{\$} IOA_A$ alors

$$Pr_C(\sigma''_{[1]})\in Tr_C \wedge Pr_B(\sigma''_{[1]})\in Tr_{IOA_{Sol}}.I_B \wedge Pr_{IOA_A}(\sigma''_{[1]})\in Tr_{IOA_A} \Rightarrow \sigma''_{[1]}\in Tr_{IOA_{Sol}}$$

Si $\sigma''_{[1]}\in O_B$, puisque $B\leq_{\$} IOA_{Sol}$ alors

$$\sigma''_{[1]}\in Tr_{IOA_{Sol}}.O_B \wedge \sigma''_{[1]}\in Tr_B \Rightarrow \sigma''_{[1]}\in Tr_{IOA_{Sol}}$$

Supposons que $\sigma''_{[k]}\in Tr_{IOA_{Sol}}$ pour $1\leq k<n$, et posons $\sigma''_{[k+1]}=\sigma''_{[k]}.u$

Si $u\in I_B$, puisque $(C\|IOA_{Sol})\leq_{\$} IOA_A$ alors

$$Pr_C(\sigma''_{[k+1]})\in Tr_C \wedge Pr_B(\sigma''_{[k+1]})\in Tr_{IOA_{Sol}}.I_B \wedge Pr_{IOA_A}(\sigma''_{[k+1]})\in Tr_{IOA_A} \Rightarrow \sigma''_{[k+1]}\in Tr_{IOA_{Sol}}$$

Si $u\in O_B$, puisque $B\leq_{\$} IOA_{Sol}$ alors

$$\sigma''_{[k+1]}\in Tr_{IOA_{Sol}}.O_B \wedge \sigma''_{[k+1]}\in Tr_B \Rightarrow \sigma''_{[k+1]}\in Tr_{IOA_{Sol}}$$

D'après le principe d'induction nous avons $\sigma''\in Tr_{IOA_{Sol}}$

Donc $\sigma''\in Tr_{(C\|IOA_{Sol}\|IOA_A)}$

Puisque $(C\|IOA_{Sol})\leq_P A$, il existe $Y_4\subseteq Tr_{(C\|IOA_{Sol}\|IOA_A)}$ tel que $Y_4\neq\emptyset$, $Y_5=Pr_B(Y_4)\subseteq OCT_{Sol}$ $(s_{0IOA_{Sol}})$ et pour tout élément $\beta\in Y_4$, σ'' est préfixe de β et $Pr_{IOA_A}(\beta)\in OCT_A(s_{0IOA_A})$

Puisque $B\leq_P Sol$, il existe $\sigma_9\in Tr_B(s_{0B})$ telle que $\sigma_9\in Y_5$

De plus il existe $\sigma_{10}\in Y_4$ telle que $\sigma_{10}\in Tr_{(C\|IOA_{Sol}\|IOA_A)}$ et $Pr_B(\sigma_{10})=\sigma_9$

Donc $\sigma_{10}\in Tr_{(C\|B\|IOA_A)}$ et $Pr_{IOA_A}(\sigma_{10})\in OCT_A(s_{0A})$.

Considérons $\sigma=\sigma_1.t\in Tr_{(C\|B)}$ telle que $t\in(I_{IOA_A}\cup O_{IOA_A})$ et $\sigma_2=Pr_{IOA_A}(\sigma)\in Tr_{IOA_A}$,

Posons $\sigma'=Pr_B(\sigma)$ et soit $|\sigma'|=n$, nous montrons par induction que $\sigma'\in Tr_{IOA_{Sol}}$

Si $\sigma'_{[1]}\in I_B$, alors il existe un préfixe σ_3 de σ tel que $Pr_B(\sigma_3)=\sigma'_{[1]}$

Puisque $(C\|IOA_{Sol})\leq_{\$} IOA_A$ alors

$$Pr_C(\sigma_3)\in Tr_C \wedge Pr_B(\sigma_3)\in Tr_{IOA_{Sol}}.I_B \wedge Pr_{IOA_A}(\sigma_3)\in Tr_{IOA_A} \Rightarrow \sigma'_{[1]}\in Tr_{IOA_{Sol}}$$

Si $\sigma'_{[1]}\in O_B$, puisque $B\leq_{\$} IOA_{Sol}$ alors $\sigma'_{[1]}\in O_B \wedge \sigma'_{[1]}\in Tr_B \Rightarrow \sigma'_{[1]}\in Tr_{IOA_{Sol}}$

Supposons que $\sigma'_{[k]}\in Tr_{IOA_{Sol}}$ pour $1\leq k<n$, et posons $\sigma'_{[k+1]}=\sigma'_{[k]}.u$

Si $u\in I_B$, alors il existe un préfixe σ_4 de σ tel que $Pr_B(\sigma_4)=\sigma'_{[k+1]}$

Puisque $(C\|IOA_{Sol})\leq_{\$} IOA_A$ alors

$$Pr_C(\sigma_4)\in Tr_C \wedge Pr_B(\sigma_4)\in Tr_{IOA_{Sol}}.I_B \wedge Pr_A(\sigma_4)\in Tr_A \Rightarrow \sigma'_{[k+1]}\in Tr_{IOA_{Sol}}$$

Si $u\in O_B$, puisque $B\leq_{\$} IOA_{Sol}$ alors

$$\sigma'_{[k+1]} \in Tr_{IOA_{Sol}}.O_B \wedge \sigma'_{[k+1]} \in Tr_B \Rightarrow \sigma'_{[k+1]} \in Tr_{IOA_{Sol}}$$

D'après le principe d'induction nous avons $\sigma' \in Tr_{IOA_{Sol}}$

Donc $\sigma \in Tr_{(C\|IOA_{Sol}\|IOA_A)}$

Vérification des propriétés (iii) dans Définition 3.12

Maintenant supposons que $MT_A((s_{0IOA_A})_{\sigma_2}) \neq \emptyset$ et considérons un $Y \in MT_A((s_{0IOA_A})_{\sigma_2})$,

Par construction de *Sol* il existe $Y_2 \subseteq Tr_{(C\|IOA_{Sol}\|IOA_A)}((s_{0(C\|IOA_{Sol}\|IOA_A)})_\sigma)$ tel que $Y_2 \neq \emptyset$, et pour tout élément $\beta \in Y_2$ on a $Pr_{IOA_A}(\beta) \in Y$

(a) $Y_2 \cap O_C \neq \emptyset$

Il existe $y \in Y_2 \cap O_C$ tel que $\sigma y \in Tr_{(C\|IOA_{Sol}\|IOA_A)}$

Puisque $(C\|B) \leq_{\S} IOA_A$ alors

$Pr_C(\sigma y) \in Tr_C \wedge Pr_B(\sigma y) \in Tr_B.I_B \wedge Pr_{IOA_A}(\sigma y) \in Tr_{IOA_A} \Rightarrow \sigma y \in Tr_{(C\|B\|IOA_A)}$

Donc $Pr_{IOA_A}(Tr_{(C\|B)}((s_{0(C\|B)})_\sigma)) \cap Y \neq \emptyset$

(b) $Y_2 \cap O_C = \emptyset$

$Y_3 = Pr_B(Y_2) \in MT_{Sol}((s_{0IOA_{Sol}})_{\sigma'})$

Puisque $B \leq_P Sol$, il existe $\sigma_5 \in Y_3$ telle que $\sigma_5 \in Tr_B((s_{0B})_{\sigma'})$

Donc il existe $\sigma_6 \in Y_2$ telle que $Pr_B(\sigma_6) = \sigma_5$

Puisque $\sigma.\sigma_6 \in Tr_{(C\|IOA_{Sol}\|IOA_A)}$ alors $Pr_C(\sigma.\sigma_6) \in Tr_C$

De plus $Pr_B(\sigma.\sigma_6) = \sigma'.\sigma_5 \in Tr_B$, donc $\sigma.\sigma_6 \in Tr_{(C\|B)}$

Donc $Pr_{IOA_A}(Tr_{(C\|B)}((s_{0(C\|B)})_\sigma)) \cap Y \neq \emptyset$

Vérification des propriétés (iv) dans Définition 3.12(iv)

Supposons qu'il existe $\sigma'' \in Tr_{(C\|B)}((s_{0(C\|B)})_\sigma)$ telle que $Pr_{IOA_A}(\sigma'') = \varepsilon$

Soit $|\sigma''| = n$, nous montrons par induction que $\sigma'' \in Tr_{IOA_{Sol}}$

Si $\sigma''_{[1]} \in I_B$, alors il existe un préfixe σ_7 de $\sigma.\sigma''$ tel que $Pr_B(\sigma_7) = \sigma'.\sigma''_{[1]}$

Puisque $(C\|IOA_{Sol}) \leq_{\S} IOA_A$ alors

$Pr_C(\sigma_7) \in Tr_C \wedge Pr_B(\sigma_7) \in Tr_{IOA_{Sol}}.I_B \wedge Pr_{IOA_A}(\sigma_7) \in Tr_{IOA_A} \Rightarrow \sigma'.\sigma''_{[1]} \in Tr_{IOA_{Sol}}$

Si $\sigma''_{[1]} \in O_B$, puisque $B \leq_{\S} IOA_{Sol}$ alors

$$\sigma'.\sigma''_{[1]} \in Tr_{IOA_{Sol}}.O_B \wedge \sigma'.\sigma''_{[1]} \in Tr_B \Rightarrow \sigma'.\sigma''_{[1]} \in Tr_{IOA_{Sol}}$$

Supposons que $\sigma'.\sigma''_{[k]} \in Tr_{IOA_{Sol}}$ pour $1 \leq k < n$,

Et posons $\sigma'.\sigma''_{[k+1]} = \sigma'.\sigma''_{[k]}.u$

Si $u \in I_B$, alors il existe un préfixe σ_8 de $\sigma.\sigma''$ tel que $Pr_B(\sigma_8) = \sigma'.\sigma''_{[k+1]}$

Puisque $(C\|IOA_{Sol}) \leq_{\S} IOA_A$ alors

$Pr_C(\sigma_8) \in Tr_C \wedge Pr_B(\sigma_8) \in Tr_{IOA_{Sol}}.I_B \wedge Pr_{IOA_A}(\sigma_8) \in Tr_{IOA_A} \Rightarrow \sigma'.\sigma''_{[k+1]} \in Tr_{IOA_{Sol}}$

Si $u \in O_B$, puisque $B \leq_{\S} IOA_{Sol}$ alors

$$\sigma'.\sigma''_{[k+1]} \in Tr_{IOA_{Sol}}.O_B \wedge \sigma'.\sigma''_{[k+1]} \in Tr_B \Rightarrow \sigma'.\sigma''_{[k+1]} \in Tr_{IOA_{Sol}}$$

D'après le principe d'induction nous avons $\sigma'.\sigma'' \in Tr_{IOA_{Sol}}$

Donc $\sigma.\sigma'' \in Tr_{(C\|IOA_{Sol}\|IOA_A)}$

Puisque $(C\|IOA_{Sol}) \leq_P A$, il existe $Y_4 \subseteq Tr_{(C\|IOA_{Sol}\|IOA_A)}((s_{0(C\|IOA_{Sol}\|IOA_A)})\sigma)$ tel que $Y_4 \neq \emptyset$, $Y_5 = Pr_B(Y_4) \subseteq OCT_{Sol}$ $((s_{0IOA_{Sol}})\sigma')$ et pour tout élément $\beta \in Y_4$, σ'' est un préfixe de β et $Pr_{IOA_A}(\beta) \in OCT_A((s_{0IOA_A})\sigma_2)$

Puisque $B \leq_P Sol$, il existe $\sigma_9 \in Tr_B((s_{0B})\sigma')$ telle que $\sigma_9 \in Y_5$

De plus il existe $\sigma_{10} \in Y_4$ telle que $\sigma.\sigma_{10} \in Tr_{(C\|IOA_{Sol}\|IOA_A)}$ et $Pr_B(\sigma_{10}) = \sigma_9$

Donc $\sigma.\sigma_{10} \in Tr_{(C\|B\|IOA_A)}$ et $Pr_{IOA_A}(\sigma_{10}) \in OCT_A((s_{0A})\sigma_2)$.

Nous concluons que $(C\|B) \leq_P A$. □

Exemple :

Dans la Figure 7.12, nous illustrons certaines solutions possibles de l'équation dans le cas de notre exemple sous formes d'automates à entrées et sorties. Toutes ces solutions sont des implantations conformes de la solution générique *Sol* obtenue précédemment et illustrée dans Figure 7.11.

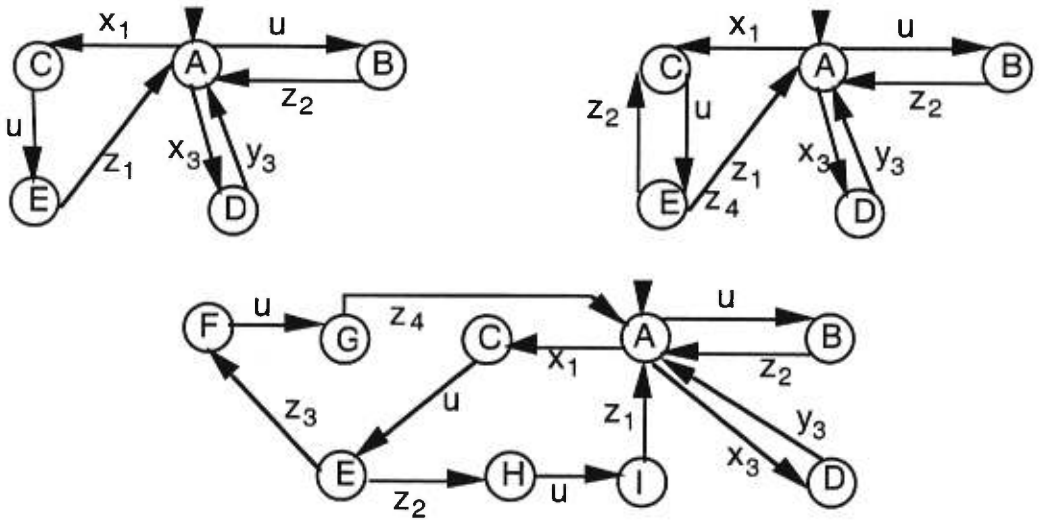


Figure 7.12 : Des implantations conformes de *Sol*.

7.4 Le cas des machines de Mealy

Les machines de Mealy peuvent être considérées comme un cas particulier d'automates à entrées et sorties. Il est donc intéressant d'adapter les résultats décrit dans ce chapitre afin de résoudre l'équation $(C\|X)\Re A$ lorsque le modèle utilisé pour les spécifications est celui des machines de Mealy. Pour être le plus général possible, nous

considérons la classe des machines de Mealy non déterministes, observables et partiellement spécifiées. La relation de conformité $\Re$ représentera l'une des relations de quasi-équivalence ou réduction.

Dans le but d'appliquer les algorithmes décrits précédemment dans ce contexte, nous signalons d'abord que l'utilisation d'une machine chaotique dans l'algorithme 7.1 est primordiale. Les machines de Mealy ne peuvent générer une sortie qu'après avoir été stimulées par une entrée. Ceci entraîne que le module à concevoir ne peut pas produire une sortie avant d'avoir été stimulé par une entrée provenant soit du contexte, soit de l'environnement. Nous utiliserons alors l'automate à entrées et sorties chaotique associé à la machine de Mealy chaotique ayant comme alphabet celui du module à concevoir. Dans ce qui suit nous décrivons l'approche utilisée. Nous transformons d'abord les spécifications de C et A afin d'obtenir les automates à entrées et sorties correspondant, notés $IOA(C)$ et $IOA(A)$ (voir chapitre 2). Puisque les machines de Mealy C et A sont observables, les automates à entrées et sorties $IOA(C)$ et $IOA(A)$ seront déterministes. Dans le cas où $\Re$ est la relation quasi-équivalence, on note A' l'automate à entrées et sorties avec traces complètes optionnelles $IOA(A)^{woct}$, c'est-à-dire, $IOA(A)^{woct}=(IOA(A), \{(s, \{\{y\}| y\in out(s)\})| s\in S_{IOA(A)}\}, \{(s, out(s))| s\in S_{IOA(A)}\})$. Par contre dans le cas où $\Re$ est la relation réduction, on note A' l'automate à entrées et sorties avec traces complètes optionnelles $IOA(A)^{red}$, c'est-à-dire, $IOA(A)^{red}=(IOA(A), \{(s, \{out(s)\})| s\in S_{IOA(A)}\}, \{(s, out(s))| s\in S_{IOA(A)}\})$. Par la suite nous utilisons Algorithme 7.2 pour résoudre l'équation $(IOA(C)\|X)\leq_{conf}A'$ sous la contrainte $I_X=(I_{IOA(A)}\backslash I_{IOA(C)})\cup(O_{IOA(C)}\backslash O_{IOA(A)})$. Nous remarquons à ce niveau que l'ensemble des entrées requis pour le module à concevoir correspond au cas de l'observabilité minimale. Dans le cas ou une solution générique *Sol* est obtenue, nous devons prouver qu'il est toujours possible de transformer *Sol* sous la forme d'une machine de Mealy.

Théorème 7.5 : Étant donnés deux machine de Mealy C et A non déterministes, observables et partiellement spécifiées. Si l'algorithme 7.2 produit un automate à entrées et sorties avec traces complètes optionnelles *Sol* lorsque il reçoit en entrée $IOA(C)$ pour la spécification du contexte, A' (égale à $IOA(A)^{woct}$ ou $IOA(A)^{red}$)pour la spécification du système entier désiré et $(I_{IOA(A)}\backslash I_{IOA(C)})\cup(O_{IOA(C)}\backslash O_{IOA(A)})$ pour l'ensemble des entrées requis alors l'ensemble des états $S_{IOA_{Sol}}$ de IOA_{Sol} vérifie les propriétés suivantes :

i - $S_{IOA_{Sol}}$ est égale à la réunion de deux sous ensembles disjoints S_{in} et S_{out},

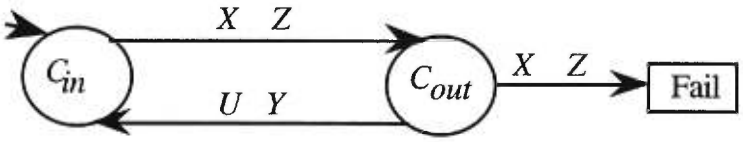
ii - $s_0 IOA_{Sol}$ appartient à S_{in},

iii - pour chaque état s dans S_{in} on a $out(s)=\emptyset$ et $entering(s)\subseteq O_{IOA_{Sol}}$,

iv - pour chaque état s dans S_{out} on a $inp(s)=\emptyset$, $out(s)\neq\emptyset$ et $entering(s)\subseteq I_{IOA_{Sol}}$.

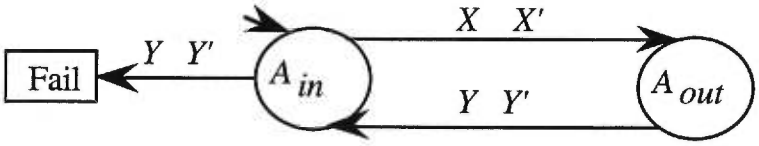
Preuve du Théorème 7.5 :

Pour la preuve du théorème, nous montrons que les propriétés sont conservées par les différents opérateurs utilisés dans l'algorithme 7.2. Afin d'obtenir la solution générique pour la réalisation sécuritaire, nous utilisons dans l'algorithme 7.1 l'opérateur de composition suivit par l'opérateur d'élimination des actions non visibles par le module à concevoir. Après complétion, l'automate $IOA(C)$ peut être représenté de la façon suivante :

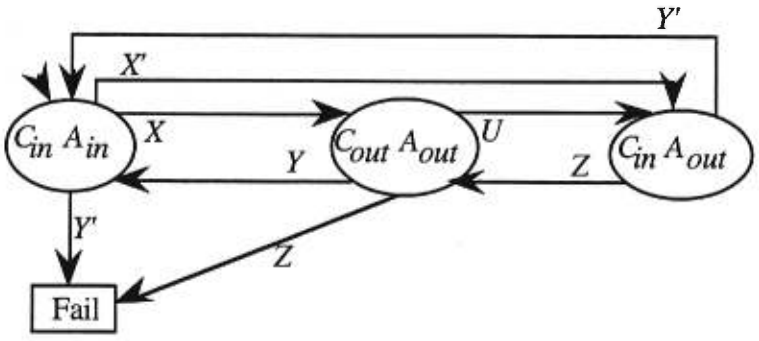


Cette représentation exprime le faite que l'ensemble des états de $IOA(C)$ est partitionné en deux sous ensembles C_{in} et C_{out}. Pour tout état s dans C_{in}, les transitions sortantes sont étiquetées par des actions dans $X\cup Z$ et les transitions entrantes sont étiquetées par des actions dans $U\cup Y$. Pour tout état s dans C_{out}, les transitions entrantes sont étiquetées par des actions dans $X\cup Z$ et les transitions sortantes sont étiquetées par des actions soit dans $U\cup Y$, où soit dans $X\cup Z$ et dans ce dernier cas elles mènent à l'état *Fail*.

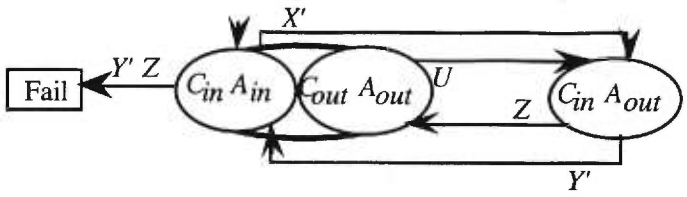
De même, après réflexion et complétion, l'automate $IOA(A)$ peut être représenté de la façon suivante :



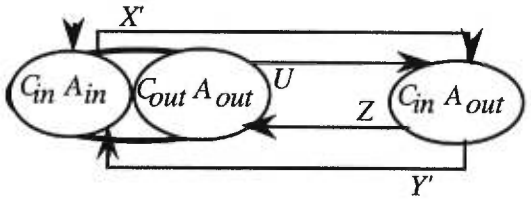
Dans le même style de représentation, la composition obtenue sera représenté par :



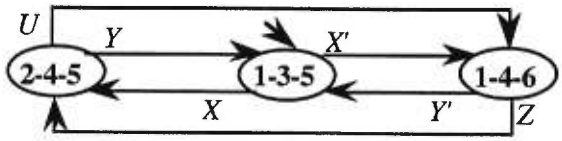
Par la suite, après élimination des actions non visibles par le module à concevoir, l'automate obtenu sera représenté par :



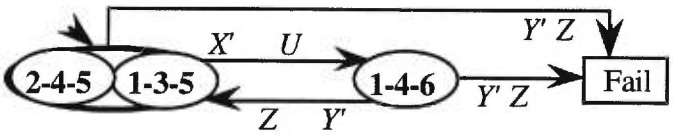
Et donc la solution générique pour la réalisation sécuritaire sera représenté par :



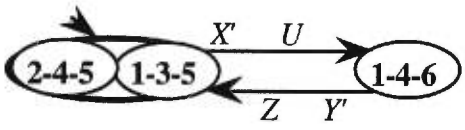
Afin de simplifier les représentations ultérieures, nous utiliserons ce qui suit : l'état C_{in} sera noté 1, l'état C_{out} sera noté 2, l'état A_{in} sera noté 3, l'état A_{out} sera noté 4, l'état C_{in} $A_{in}C_{out}A_{out}$ sera noté 5 et l'état C_{in} A_{out} sera noté 6. Par la suite dans l'algorithme 7.2, la composition obtenue sera représenté par :



Après l'application des étapes 2, 3 et 4, l'automate obtenu sera représenté par :



Si l'algorithme 7.2 produit une solution Sol alors elle sera représentée par :



Ceci indique bien que l'ensemble des états de la solution générique est partitionnée en deux sous ensembles. Les propriétés (i), (ii) et (iii) sont évidentes à partir de la représentation de la solution. Pour la propriété (iv), il faut s'assurer que $out(s)\neq\emptyset$. Or si un tel état existait il serait silencieux, ceci est impossible car il aurait été éliminé par l'algorithme.

Donc la solution générique vérifie bien les propriétés du théorème. □

D'après le théorème 7.5, on peut construire une machine de Mealy $FSM_{Sol}=(S, I_{IOA_{Sol}}, O_{IOA_{Sol}}, h, s_0)$. L'ensemble des états S de FSM_{Sol} est égal à S_{in} et $s_0=s_{0IOA_{Sol}}$. Pour chaque état s_i de FSM_{Sol}, il existe une transition $s_i \rightarrow v/w \rightarrow s_k$ dans FSM_{Sol} si et seulement si il existent les transitions $s_i \rightarrow v \rightarrow s_j$ et $s_j \rightarrow w \rightarrow s_k$ dans IOA_{Sol}. La machine de Mealy est donc une solution de l'équation $(C||X)\Re A$. De plus, pour toute machine de Mealy D solution de l'équation $(C||X)\Re A$, l'automate à entrées et sorties correspondant $IOA(D)$ est une solution de l'équation $(IOA(C)||X) \leq_{conf} A'$. L'ensemble des solutions de l'équation $(C||X)\Re A$ coïncide donc avec l'ensemble des machines de Mealy quasi-équivalentes à l'une des machines de Mealy dérivée à partir d'un automate à entrées et sorties qui est une implantation conforme de *Sol* et inclus dans IOA_{Sol}.

En particulier dans le cas où les machines de Mealy C et A sont complètement spécifiées, la relation quasi-équivalence se réduit à la relation d'équivalence, et nous obtenons l'ensemble des solutions de l'équation pour les machines de Mealy complètement spécifiées en utilisant la même approche. Pour l'exemple 1 de la section 5.4 du chapitre 5, nous obtenons qu'il n'y a pas de solution. Par contre, pour l'exemple 2 de la section 5.4 du chapitre 5, nous obtenons une solution générique *Sol*. L'automate à entrées et sorties IOA_{Sol} décrit la machine chaotique de la Figure 7.13,

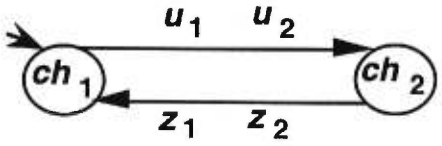


Figure 7.13 : La machine chaotique IOA_{Sol}

$MT_{Sol}=\{(ch, \emptyset)\}$ et $OCT_{Sol}=\{(ch, \{u_1z_1(u_2z_1)^*u_2z_2, u_1z_2(u_2z_2)^*u_2z_1\})\}$. Nous constatons effectivement que les résultats décrits dans l'exemple 2 sont confirmés.

On peut donc dire que cette approche permet une caractérisation totale de l'ensemble des solutions pour le problème de construction de sous-modules dans le modèle des machines de Mealy. Ceci indique que la relation de réduction est non suffisante pour l'extraction des différentes solutions à partir de l'ensemble des comportements permis dans le cas où la relation de conformité utilisée dans l'équation est soit la relation d'équivalence de traces ou soit la relation quasi équivalence [Watanabe 93][Aziz 95][Petrenko 98]. Ceci est du au fait qu'après réception d'une entrée dans un état s de la solution générique, on doit imposer qu'au moins une trace, de longueur supérieure à 1, parmi un ensemble de traces est présente dans une machine de Mealy M après réception de la même entrée dans l'état

correspondant à s afin de garantir que M est une solution. Cette approche offre aussi une méthode uniforme pour la résolution de l'équation pour les différentes classes de machines de Mealy ainsi que pour différentes relations de conformité.

CHAPITRE 8

LA CONSTRUCTION DANS LE MODÈLE DES AUTOMATES À ENTRÉES ET SORTIES TEMPORISÉS

Les programmes temps-réel tels que les contrôleurs des systèmes de navigation, les systèmes d'exploitation temps-réel, les logiciels de commutation et les contrôleurs de systèmes de production sont fondamentalement réactifs, et leurs interactions avec l'environnement doivent satisfaire des contraintes temporels. D'autres systèmes où la notion explicite du temps joue un rôle important sont les systèmes de communications et particulièrement les protocoles de communication; la performance de tels système dépend de façon vitale de la valeur des temporisateurs qui contrôlent la retransmission des messages. Le fonctionnement correct de ces systèmes ne dépends pas seulement d'une consistance logique en terme de séquences d'événements, il faut en plus satisfaire des contraintes temporels. Des modèles temps-réel ont été introduits pour la spécification et la vérification des systèmes temps-réel [Alur 94]. Ces modèles sont obtenus à partir du modèle des automates en introduisant le temps par l'intermédiaire d'horloges et de contraintes sur ces horloges.

Dans [Maler 95], les auteurs montrent que le problème de la synthèse de contrôleurs est solvable lorsque les spécifications du système à contrôler ainsi que celle du comportement désiré sont données sous la forme d'automates temporisés. La solution est obtenue par la résolution d'équations grâce à la technique du point fixe. Dans ce travail, les auteurs supposent que le contrôleur peut observer précisément la configuration global du système à contrôler. Cependant, dans des situations réelles la configuration du système ne peut être observée que partiellement, et le contrôleur doit fonctionner avec une certaine incertitude sur l'état du système.

Dans ce chapitre, nous généralisons le travail du chapitre 7 au cas où les

spécifications sont données sous la forme d'automates à entrées et sorties temporisés. La relation de conformité utilisée est la réalisation sécuritaire. Nous considérons le cas où le module à concevoir observe précisément la configuration global de la partie existante ainsi que celui où l'observation est partielle.

8.1 Définitions et propriétés

Dans cette partie, nous présentons un modèle pour les systèmes temps-réel ayant un nombre fini d'états de contrôle, une variante des automates temporisés [Alur 94]. Ce modèle est obtenu à partir des automates à entrées et sorties en introduisant le temps par l'intermédiaire d'horloges et de contraintes sur ces horloges.

Intuitivement, les horloges sont des sortes de chronomètres qui avancent à la même vitesse que le temps et que l'on peut remettre à zéro indépendamment les unes des autres. Au départ toutes les horloges sont initialisées à zéro. Une horloge peut être remise à zéro lors de l'exécution d'une transition. À n'importe quel moment, la valeur indiquée par une horloge est égale au temps écoulé depuis la dernière fois qu'elle a été remise à zéro.

Les contraintes d'horloges sont les formules définies par la grammaire :

$$\varphi := x \sim c \mid x - y \sim c \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi$$

où x et y sont des horloges dans un ensemble X, $\sim \in \{<, \leq, =, \geq, >\}$et c est une constante dans l'ensemble $\mathbb{N}$ des entiers positifs. L'ensemble des contraintes d'horloges sur un ensemble d'horloges X est noté $\Phi(X)$. Une contrainte d'horloge est dite positive si elle peut s'écrire sans $\neg$ et sans $\vee$.

Une valuation d'horloge ν sur un ensemble X d'horloges associe une valeur réelle à chaque horloge; c'est une application de X vers l'ensemble des réels positifs $\mathbb{R}^+$. Étant donné un ordre sur X, si d représente le nombre d'éléments de X, on note $\nu=(\nu_1, ..., \nu_d)$, la valuation d'horloge $\nu : x_1 \mapsto \nu_1, ..., x_n \mapsto \nu_d$. Soit $\nu=(\nu_1, ..., \nu_d)$ une valuation d'horloge et $\delta \in \mathbb{R}^+$, $\nu+\delta$ représente la valuation d'horloge $(\nu_1+\delta, ..., \nu_d+\delta)$. Pour $Y \subseteq X$, $\nu[Y:=0]$ représente la valuation d'horloge définie comme suit :

$$\nu[Y:=0](x) = \begin{cases} 0 \text{ si } x \in Y \\ \nu(x) \text{ si } x \notin Y \end{cases}$$

Définition 8.1 [Maler 95]: À chaque constante entière positive k , on associe trois sous ensembles de $2^{\mathbb{R}^d}$ où d est une constante entière positive:

H_k : contient les ensembles ayant l'une des formes suivante : $\mathbb{R}^d$, $\emptyset$, $\{(v_1, ..., v_i, ..., v_d) \in \mathbb{R}^d : v_i \sim c\}$, $\{(v_1, ..., v_i, ..., v_j, ..., v_d) \in \mathbb{R}^d : v_i - v_j \sim c\}$, pour $\sim \in \{<, \leq, >, \geq\}$ et $c \in \{0, ..., k\}$.

$H_k^{\cap}$: contient les ensembles convexes formés par l'intersection d'éléments de H_k.

H_k^* : contient les k-polyhèdres où un k-polyhèdre est un ensemble obtenu à partir des éléments de H_k en utilisant les opérateurs ensemblistes réunion, intersection et complément.

Pour chaque k, H_k^* a un nombre fini d'éléments, chacun pouvant s'écrire comme une union finie d'ensembles convexes. Dans la définition précédante, d représente le nombre d'horloges et k représente une borne supérieure sur les constantes présentes dans les contraintes d'horloges.

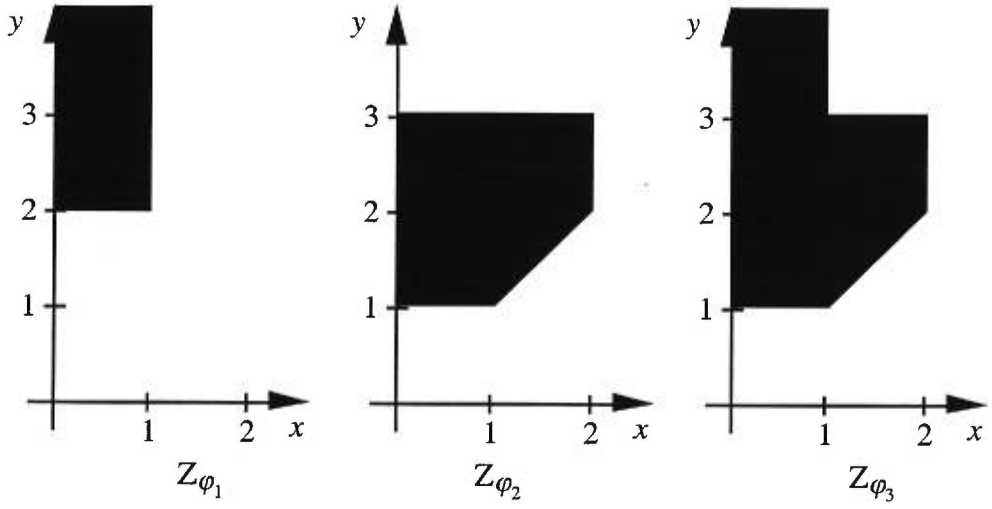


Figure 8.1 : Zones associées à différentes contraintes

Une valuation d'horloge ν satisfait une contrainte d'horloge φ sur un ensemble d'horloges X, on note $\nu \models \varphi$, si et seulement si $\varphi(\nu)$ a la valeur vraie. Une contrainte d'horloge φ défini un sous ensemble Z_φ de $(\mathbb{R}^+)^n$ appelé une zone : $Z_\varphi = \{\nu \text{ tel que } \nu \models \varphi\}$.

La figure 8.1 représente les zones définies par les contraintes $\varphi_1 = (x \leq 1) \wedge (y \geq 2)$, $\varphi_2 = (y \geq 1) \wedge (y \leq 3) \wedge (x \leq 2) \wedge (y - x \geq 0)$ et $\varphi_3 = ((x \leq 1) \wedge (y \geq 2)) \vee ((y \geq 1) \wedge (y \leq 3) \wedge (x \leq 2) \wedge (y - x \geq 0))$. Les deux premières zones sont convexes, pas la troisième; mais elle est la réunion des deux premières zones.

Définition 8.2 : Un automates à entrées et sorties temporisés (*TIOA* : Timed Input Output

Automaton) A, est un tuple (S_A, I_A, O_A, X_A, M_A, T_A, s_{0A}) où :

- S_A est un ensemble fini non vides d'états,
- I_A est un ensemble fini non vides d'entrées,
- O_A est un ensemble fini non vides de sortie avec $I_A \cap O_A = \emptyset$,
- X_A est un ensemble fini d'horloges,
- M_A est une application de S_A vers $\Phi(X_A)$, elle associe à chaque état s une contrainte d'horloge appelée l'invariant de l'état et notée $Inv(s)$,
- $T_A \subseteq S_A \times (I_A \cup O_A) \times 2^{X_A} \times \Phi(X_A) \times S_A$ est un ensemble de transitions. Un élément (s, u, λ, φ, s')$\in T_A$ représente une transition à partir de l'état s vers l'état s' étiquetée par u, $\lambda \subseteq X_A$ est l'ensemble des horloges à remettre à zéro et φ est une contrainte sur les horloges dans X_A qui spécifie quand la transition est exécutable,
- s_{0A} est l'état initial.

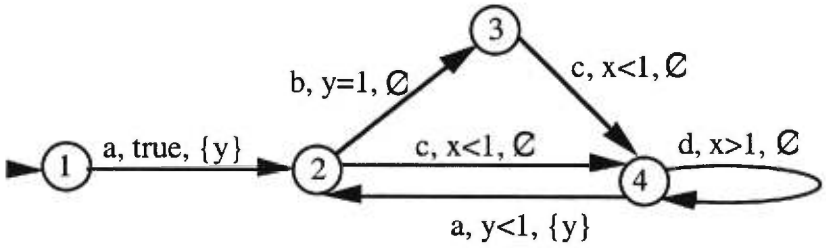


Figure 8.2 : Représentation graphique d'un automate à entrées et sorties temporisé

La figure 8.2 illustre la représentation graphique de l'automate à entrées et sorties temporisé A=({1, 2, 3, 4}, {a, b}, {c, d}, {x, y}, M_A, {(1, a, {y}, *true*, 2), (2, b, ∅, y=1, 3), (2, c, ∅, x<1, 4), (3, c, ∅, x<1, 4), (4, a, {y}, y<1, 2), (4, d, ∅, x>1, 4)}, 1} avec $M_A(s)$=*true* pour tout $s\in$ {1, 2, 3, 4}.

Alors que les transitions sont instantanées, le temps peut s'écouler dans un état, c'est à dire l'automate peut rester dans l'état et laisser les valeurs des horloges augmenter de façon uniforme aussi longtemps que l'invariant est satisfait [Labroue 98]. Puisque dans le modèle des automates à entrées et sorties, une distinction claire existe entre les actions contrôlées par l'automate -les sorties- et celles contrôlées par l'environnement -les entrées-, nous assumons qu'au moins une action étiquetée par une sortie est exécutable juste avant la violation de l'invariant. Dans la pratique, les invariants associés aux états sont fermés à droite, c'est à dire que pour tout instant t, si il existe $\varepsilon>0$ tel que l'invariant est vrai à t-δ pour tout $\varepsilon>\delta>0$ alors l'invariant est vrai à t. Dans ce cas, si l'invariant associé à un état est vrai à un instant t et faux pour tout instant $t+\delta$ pour tout $\delta>0$ alors au moins une action étiquetée par une sortie

doit être exécutable à l'instant t. Les actions étiquetées par une sortie à l'instant t son considérées urgentes, l'exécution de l'une d'elles doit être forcée à cet instant afin de quitter l'état avant la violation de l'invariant.

Si pour chaque $s \in S_A$, tout $u \in I_A$ et toute valuation d'horloge ν satisfaisant $Inv(s)$ il existe $(s, u, \lambda, \varphi, s') \in T_A$ telle que ν satisfait aussi φ, alors A est complètement spécifié -input-enabled-; sinon A est partiellement spécifié. Un automate à entrées et sorties temporisé A est non déterministe si ils existent $(s, u, \lambda, \varphi, s') \in T_A$, $(s, u, \lambda', \varphi', s'') \in T_A$ avec $s' \neq s''$ et une valuation d'horloge ν qui satisfait simultanément $Inv(s)$, φ et φ' pour au moins un état s et une action u; sinon A est déterministe. La forme complétée d'un automate à entrées et sorties temporisé A, notée $Tief(A)$, est obtenue de la façon suivante : pour chaque état $s_1 \in S_A$ et chaque entrée $x_1 \in I_A$, soit $Ctrans(s_1, x_1) = \{\varphi \mid \exists (s_1, x_1, \lambda, \varphi, s') \in T_A\}$ l'ensemble des contraintes d'horloges associées aux transitions ayant pour source l'état s_1 et étiquetées par x_1, s'il existe au moins une valuation d'horloge qui satisfait $Inv(s_1)$ et ne satisfait aucune contrainte dans $Ctrans(s_1, x_1)$ alors on ajoute à l'automate la transition $(s_1, x_1, \emptyset, Inv(s_1) \wedge \neg(\bigvee_{\varphi \in Ctrans(s_1, x_1)} \varphi), Fail_A)$ où $Fail_A$ est un état spécial.

Considérons la séquence $\sigma = (a_1, t_1)...(a_k, t_k) \in ((I_A \cup O_A) \times \mathbb{R})^*$ avec $t_1, ..., t_k$ une suite croissante dans $\mathbb{R}^+$, et soient $\nu_1, ..., \nu_k$ des valuations d'horloges définies par :

$\nu_1(x) = t_1$ pour tout $x \in X_A$ et

$\nu_{i+1} = \nu_i[\lambda_i := 0] + (t_{i+1} - t_i)$ pour $k-1 \geq i \geq 1$,

la séquence σ est une trace temporisée à partir de l'état s, si ils existent des états $s_1, ..., s_{k+1} \in S_A$ tels que $s_1 = s$, $(s_i, a_i, \lambda_i, \varphi_i, s_{i+1}) \in T_A$, et ν_i satisfait simultanément $Inv(s_i)$ et φ_i pour $i = 1,...,k$. L'ensemble des traces temporisées à partir de l'état s est noté $Ttr_A(s)$ et on le note Ttr_A si $s = s_{0A}$. Un état s' d'un automate à entrées et sorties temporisé A est accessible à partir de l'état s s'il existe une trace temporisée $\sigma \in Ttr_A(s)$ telle que l'exécution de σ à partir de s mène à s'. Si s est l'état initial de A alors s' est accessible.

La composante connexe contenant l'état initial, notée $CC(A)$, est l'automate à entrées et sorties temporisé $CC(A) = (S_C, I_C, O_C, X_C, M_C, T_C, s_{0C})$ tel que $S_C = \{s \in S_A \mid s$ est accessible$\}$, $I_C = I_A$, $O_C = O_A$, $X_C = X_A$, $M_C = M_A|S_C$, $T_C = \{(s, u, \lambda, \varphi, s') \in T_A \mid s \in S_C\}$ et $s_{0C} = s_{0A}$, la notation $M_A|S_C$ représente la restriction de M_A à l'ensemble S_C. Si $A = CC(A)$ alors A est initialement connecté.

8.2 Les opérateurs et relations de conformité

8.2.1 La composition

Définition 8.3 : Considérons deux automates à entrées et sorties temporisés $A_1=(S_{A_1}, I_{A_1}, O_{A_1}, X_{A_1}, M_{A_1}, T_{A_1}, s_{o1})$ et $A_2=(S_{A_2}, I_{A_2}, O_{A_2}, X_{A_2}, M_{A_2}, T_{A_2}, s_{o2})$ tels que $O_{A_1}\cap O_{A_2}=\emptyset$ et $X_{A_1}\cap X_{A_2}=\emptyset$. La composition de A_1 et A_2 notée $A_1||A_2$, est par définition égale à la composante connexe de l'automate à entrées et sorties temporisés $A=(S_A, I_A, O_A, X_A, M_A, T_A, s_{oA})$ où :

- $S_A=S_{A_1}\times S_{A_2}$,
- $I_A=(I_{A_1}\cup I_{A_2})\setminus(O_{A_1}\cup O_{A_2})$,
- $O_A=O_{A_1}\cup O_{A_2}$,
- $X_A=X_{A_1}\cup X_{A_2}$,
- $M_A(s_1, s_2)=M_{A_1}(s_1)\wedge M_{A_2}(s_2)$
- $((s_1, s_2), u, \lambda_1\cup\lambda_2, \varphi_1\wedge\varphi_2, (s_1', s_2'))\in T_A$ si et seulement si $\varphi_1\wedge\varphi_2\wedge Inv((s_1, s_2))\neq$false et pour tout $i\in\{1, 2\}$, si $u\in(I_{A_i}\cup O_{A_i})$ alors $(s_i, u, \lambda_i, \varphi_i, s_i')\in T_{A_i}$, sinon $s_i=s_i'$, $\lambda_i=\emptyset$ et $\varphi_i=$true,
- $s_{oA}=(s_{o1}, s_{o2})$.

La composition d'automates à entrées et sorties temporisés est commutative et associative. Elle permet qu'un nombre quelconque d'automates à entrées et sorties temporisés acceptent la même entrée simultanément.

Nous définissons une propriété de sécurité afin de formaliser la non existence de réceptions non spécifiées dans la composition A d'une collection d'automates à entrées et sorties temporisés $(A_i=(S_{A_i}, I_{A_i}, O_{A_i}, X_{A_i}, M_{A_i}, T_{A_i}, s_{oi}))_{1\leq i\leq n}$. On note ε le mot vide.

Définition 8.4 : Étant donnée une collection d'automates à entrées et sorties temporisés $(A_i=(S_{A_i}, I_{A_i}, O_{A_i}, X_{A_i}, M_{A_i}, T_{A_i}, s_{oi}))_{1\leq i\leq n}$, la composition $A=A_1||A_2||...||A_n$ est sécuritaire, noté $\mathbb{S}(A)$, si et seulement si tout mot t dans $((I_A\cup O_A)\times\mathbb{R})^*$ tel que $Pr_{A_i}(t)\in Tr_{A_i}.((I_{A_i}\times\mathbb{R})\cup\{\varepsilon\})$ pour tout i, est une trace de A (c'est à dire $t\in Ttr_A$).

8.2.2 Les relations de conformité

8.2.2.1 La réalisation sécuritaire

Définition 8.5 : Considérons un automate à entrées et sorties temporisé A et un automate à entrées et sorties temporisé $B=B_1||B_2||...||B_n$ tel que $I_A=I_B$. B est une réalisation sécuritaire de A, noté $B\leq_{\mathbb{S}}A$, si et seulement si pour tout automate à entrées et sorties temporisé E, tel que $I_E=O_A$ et $O_E=I_A$, $\mathbb{S}(E||A)$ implique $\mathbb{S}(E||B_1||B_2||...||B_n)$.

La Définition 8.5 capture le faite que pour tout environnement E, ayant les même ensembles d'entrées et de sorties que A, si la composition de A et E est sécuritaire alors la composition de $B_1, B_2, ..., B_n$ et E doit aussi être sécuritaire, c'est-à-dire, dans tout état accessible de la composition $E\|B_1\|B_2\|...\|B_n$, il n'existe pas de réception non spécifiée.

La réflexion $\tilde{A}$ d'un automate à entrées et sorties temporisé A représente l'environnement temporisé le plus libéral par rapport aux sorties et le moins défini par rapport aux entrées dans lequel A est sécuritaire, c'est-à-dire, $\tilde{A}$ doit accepter toutes les sorties produite par A et rien de plus, et produire toutes les entrées acceptées par A. Dans le cas des automates à entrées et sorties temporisés, la classe déterministe est strictement incluse dans la classe non-déterministe [Alur 94a]. Pour travailler dans le cas le plus général, nous devons donc considérer la construction de la réflexion d'un automate à entrées et sorties temporisé non-déterministe. Ceci implique que tout état s de $\tilde{A}$ doit avoir les propriétés suivantes :

- l'ensemble des sorties de s doit être égale à l'intersection des ensembles d'entrées de tous les états de A qui sont accessibles par une trace temporisé menant à s.
- l'ensemble des entrées de s doit être égale à l'union des ensembles de sorties de tous les états de A qui sont accessibles par une trace temporisé menant à s.

Étant donné que le problème dans la construction de la réflexion d'un automate à entrées et sorties temporisé non-déterministe provient du non-déterminisme, nous illustrons la nécessite des propriétés précédantes à travers la construction de la réflexion de l'automate à entrées et sorties non-déterministe A de la figure 8.3 avec $I_A=\{x_1, x_2, x_3\}$ et $O_A=\{y_1, y_2\}$.

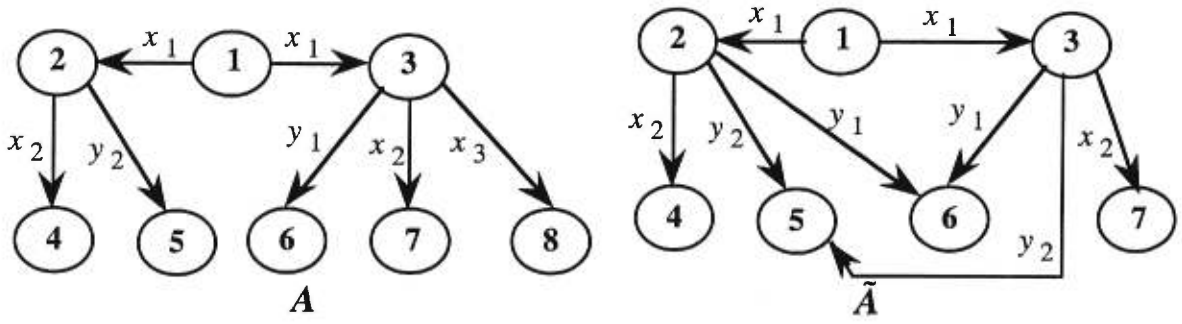


Figure 8.3 : L'automate à entrées et sorties non-déterministe A et sa réflexion $\tilde{A}$

L'environnement le plus libéral par rapport aux sorties et le moins défini par rapport aux entrées dans lequel le système décrit par A est sécuritaire (figure 8.3), pourra dans son état initial offrir au système l'entrée x_1. Après réception de cette entrée, le système transitera à l'un des états 2 ou 3. Comme l'environnement ne connaît pas l'état réel du système, il ne

devra pas offrir l'entrée x_3 car sa réception n'est pas spécifiée dans l'état 2. Il devra offrir seulement l'entrée x_2, c'est-à-dire, l'intersection des ensembles d'entrées présentes dans les états 2 et 3. Par contre, l'environnement devra s'attendre à recevoir la sortie y_1 ainsi que la sortie y_2, c'est-à-dire, l'union des ensembles de sorties présentes dans les états 2 et 3.

Pour la construction de $\tilde{A}$, nous utilisons les deux fonctions *RestrictOutputs* et *AugmentInputs* décrites ci-dessous. La première fonction prends un automate à entrées et sorties temporisé A en argument et produit un automate à entrées et sorties temporisé *RestrictOutputs*(A) dans lequel tous les états accessibles par une trace temporisée commune ont le même ensemble de sortie (l'intersection de leurs ensembles de sorties). La deuxième fonction prends un automate à entrées et sorties temporisé A en argument et produit un automate à entrées et sorties temporisé *AugmentInputs*(A) dans lequel tous les états accessibles par une trace temporisée commune ont le même ensemble d'entrées (l'union de leurs ensembles d'entrées). Dans le cas d'un automate à entrées et sorties temporisé déterministe A, on obtient *RestrictOutputs*(A)=*AugmentInputs*(A)=A.

Pour deux chemins ayant la même trace non temporisée d'un automate à entrées et sorties temporisé A, on appelle partie commune, le chemin ayant exactement pour traces temporisées celles qui sont communes aux deux chemins. Les deux fonctions *RestrictOutputs* et *AugmentInputs* utilisent la fonction *SépareChemins*. Cette fonction a deux paramètres, un automate à entrées et sorties temporisé A et un de ses état s. Elle commence par trouver tous les chemins qui mènent à s. Par la suite chaque chemin P menant à un état s' différent de s et ayant la même trace non temporisée qu'un chemin P' menant à s est décomposé en deux chemins. Le premier chemin représente la partie commune de P et P'. Le deuxième chemin représente la partie complémentaire de P par rapport à P'. Les pseudo-codes associés aux fonctions *SépareChemins*, *RestrictOutputs* et *AugmentInputs* sont les suivants :

SépareChemins*(*TIOA A*, *État s*) : *TIOA

TEMP=A;
Éliminer à partir de *TEMP* tous les états à partir desquels on ne peut pas atteindre s;
$I_{TEMP}=I_A \cup O_A$;
$O_{TEMP}=\emptyset$;
Renommer toutes les horloges dans *TEMP*;
TEMP=*TEMP* $\|A$;
Éliminer de *TEMP* tous les états à partir desquels on ne peut pas atteindre un état de la

forme (s, s') avec $s' \neq s$;
Ajouter à A les états (s_1, s_2) de *TEMP* tels que $s_1 \neq s_2$;
POUR chaque état (s_1, s_2) de *TEMP* **FAIRE**
SI $s_1 \neq s_2$ **ALORS**
POUR chaque transition $((s_1, s_2), u, \lambda, \varphi, (s_3, s_4))$ dans *TEMP* **FAIRE**
SI $s_3 \neq s_4$ **ALORS**
Ajouter la transition $((s_1, s_2), u, \lambda, \varphi, (s_3, s_4))$ à A;
SINON
Ajouter la transition $((s_1, s_2), u, \lambda, \varphi, s_3)$ à A;
Ajouter la transition $((s_1, s_2), u, \lambda', \varphi', s_4)$ à A pour compléter la transition correspondante $(s_2, u, \lambda', \varphi'', s_4)$ dans A avec $\varphi' = \varphi'' \wedge \neg\varphi$;
POUR chaque transition $(s_2, u', \lambda', \varphi', s')$ dans A **FAIRE**
SI u' n'est pas étiquette d'une transition sortante de (s_1, s_2) dans A **ALORS**
Ajouter la transition $((s_1, s_2), u', \lambda', \varphi', s')$ dans A;
SINON
POUR chaque transition $((s_1, s_2), u, \lambda, \varphi, (s_3, s_4))$ dans *TEMP* **FAIRE**
SI $s_3 \neq s_4$ **ALORS**
Ajouter la transition $(s_1, u, \lambda, \varphi, (s_3, s_4))$ dans A;
Remplacer la transition $(s_1, u, \lambda', \varphi', s_4)$ dans A par $(s_1, u, \lambda', \varphi' \wedge \neg\varphi, s_4)$;

RestrictOutputs(TIOA A) : TIOA
POUR chaque état s dans A **FAIRE**
A=*SépareChemins*(A, s);
POUR chaque $u \in O_A$ **FAIRE**
SI il n'existe pas de transition sortante de s étiquetée par u **ALORS**
POUR chaque état (s, s_2) de A **FAIRE**
Éliminer de A toutes les transitions sortantes de (s, s_2) et étiquetée par u;
SINON
PresConst $:= \{\varphi \mid (s, u, \lambda, \varphi, s')$ dans $A\}$;
$\varphi'' := (\bigvee_{\varphi \in PresConst} \varphi)$;
Remplacer chaque transition $((s, s_2), u, \lambda', \varphi', s')$ dans A par $((s, s_2), u, \lambda', \varphi'' \wedge \varphi', s')$.

AugmentInputs(TIOA A) : TIOA

POUR chaque état s dans A **FAIRE**
  A=*SépareChemins*(A, s);
  POUR chaque état (s, s_2) de A **FAIRE**
    POUR chaque transition $(s, u, \lambda, \varphi, s_3)$ dans A **FAIRE**
      SI $u \in I_A$ **ALORS**
        $PresConst := \{\varphi' \mid ((s, s_2), u, \lambda', \varphi', s')$ dans $A\}$;
        $\varphi'' := \varphi \wedge \neg(\bigvee_{\varphi' \in PresConst} \varphi')$;
        Ajouter la transition $((s, s_2), u, \lambda, \varphi'', s_3)$ dans A;

Définition 8.6 : La réflexion d'un automate à entrées et sorties temporisé $A=(S_A, I_A, O_A, X_A, M_A, T_A, s_{oA})$ est l'automate à entrées et sorties temporisé $\tilde{A}$=*AugmentInputs*(*RestrictOutputs*((S_A, $I_{\tilde{A}}$, $O_{\tilde{A}}$, X_A, M_A, T_A, s_{oA}))) tel que $I_{\tilde{A}}=O_A$ et $O_{\tilde{A}}=I_A$.

Lemme 8.1: Pour deux automates à entrées et sorties temporisés A et $B=B_1\|B_2\|...\|B_n$ tels que $I_A=I_B$, les propositions suivante sont équivalentes :

i - $\mathbb{S}(\tilde{A}\|B_1\|B_2\|...\|B_n)$,

ii - pour tout automate à entrées et sorties temporisé E, tel que $I_E=O_A$ et $O_E=I_A$,

$$\mathbb{S}(E\|A) \Rightarrow \mathbb{S}(E\|B_1\|B_2\|...\|B_n).$$

La preuve du lemme 8.1 est similaire à celle de Proposition 3.2.

8.2.2.2 La réalisation T-sécuritaire

Définition 8.7 : Pour deux automates à entrées et sorties temporisés A et $B=B_1\|B_2\|...\|B_n$ tels que $I_A=I_B$, on dit que B est une réalisation T-sécuritaire de A, et on note $B \leq_{\mathbb{T}\mathbb{S}} A$, si et seulement si

i - $B \leq_{\mathbb{S}} A$,

ii - pour tout état $(s, s_1, s_2, ..., s_n)$ dans $\tilde{A}\|B_1\|B_2\|...\|B_n$ $Inv(s_1, s_2, ..., s_n)$ implique $Inv(s)$.

La Définition 8.7 exige que $B_1\|B_2\|...\|B_n$ soit une réalisation sécuritaire de A, de plus, pour chaque état $(s_1, s_2, ..., s_n)$ dans $B_1\|B_2\|...\|B_n$ tel que $(s, s_1, s_2, ..., s_n)$ est dans $\tilde{A}\|B_1\|B_2\|...\|B_n$ l'invariant $Inv((s_1, s_2, ..., s_n))$ implique $Inv(s)$.

8.3 Transformations pour les automates à entrées et sorties

temporisés

8.3.1 Élimination des transitions invisibles d'un automate à entrées et sorties temporisé

Lorsqu'une transition est étiquetée par une action invisible, on l'appelle une τ-transition. Par exemple, une action qui est interne dans une composition. Il est bien connu que de telles transitions n'augmentent pas le pouvoir expressif des automates non-temporisés. Dans le cas des automates temporisés, la situation est différente. Les τ-transitions avec remise à zéro d'horloges augmentent strictement le pouvoir expressif des automates temporisés [Bérard 98].

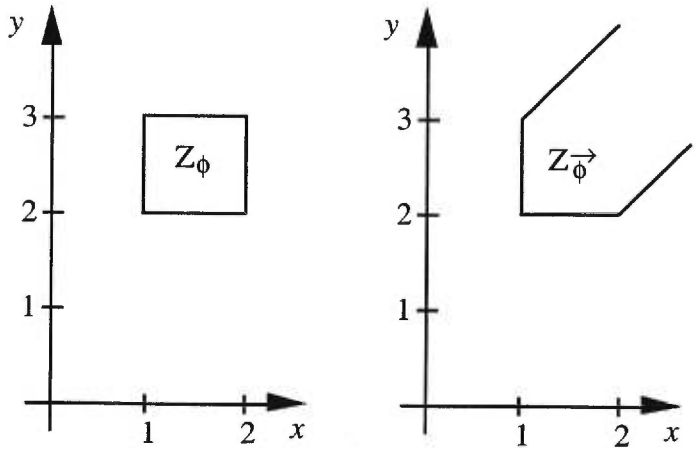


Figure 8.4 : Les zones Z_ϕ et $Z_{\overrightarrow{\phi}}$ associées aux contraintes d'horloges ϕ et $\overrightarrow{\phi}$

Dans le cas des τ-transitions sans remise à zéro d'horloges, c'est-à-dire les τ-transitions dans lesquelles aucune horloge n'est remise à zéro, nous trouvons dans [Bérard 96] un algorithme pour construire à partir d'un automate à entrées et sorties temporisé, un automate à entrées et sorties temporisé équivalent ne contenant aucune τ-transitions. Cet algorithme se compose de deux parties. Dans la première partie, les cycles formés par des τ-transitions sont éliminés. Dans la deuxième partie, toutes les τ-transitions restantes sont éliminés. L'intérêt d'une τ-transition sans remise à zéro d'horloges est de vérifier qu'une certaine contrainte d'horloges présente dans la transition est satisfaite à un instant donné. L'idée principale dans l'élimination d'une τ-transition sans remise à zéro d'horloges est de déplacer la vérification de la contrainte soit vers la transition précédante soit vers la transition suivante. Ceci est réalisé par l'utilisation de la clôture par le passé (backward closure) ou de la clôture dans le future (forward closure) et par l'ajout de nouvelles horloges.

Définition 8.8 [Bérard 98]: Soit ϕ une contrainte d'horloges. La clôture dans le future de ϕ, notée $\overrightarrow{\phi}$, est une formule qui est satisfaite par une valuation d'horloges ν si ϕ est satisfaite par la valuation d'horloges ν-δ pour un certain $\delta \geq 0$:

$$\nu \models \overrightarrow{\phi} \text{ si } \exists \delta \geq 0, \nu\text{-}\delta \models \phi$$

Définition 8.9 [Bérard 98]: Soient ϕ une contrainte d'horloges et λ un ensemble d'horloges. La clôture par le passé de ϕ par rapport à λ, notée $\overleftarrow{\phi}^{\lambda}$, est une formule qui est satisfaite par une valuation d'horloges ν si ϕ est satisfaite par la valuation d'horloges $\nu[\lambda:=0]+\delta$ pour un certain $\delta \geq 0$:

$$\nu \models \overleftarrow{\phi}^{\lambda} \text{ si } \exists \delta \geq 0, \nu[\lambda:=0]+\delta \models \phi$$

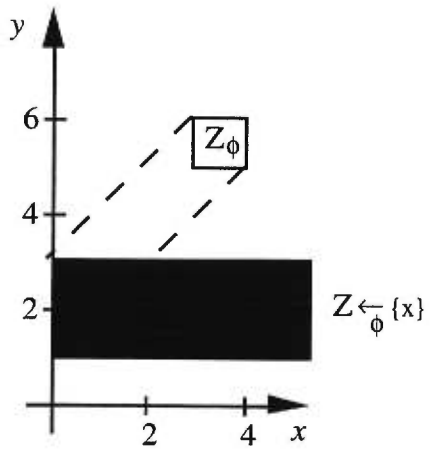


Figure 8.5 : Les zones Z_ϕ et $Z_{\overleftarrow{\phi}\{x\}}$ associées aux contraintes d'horloges ϕ et $\overleftarrow{\phi}^{\{x\}}$

Lemme 8.2 [Bérard 98]**:** Soient ϕ une contrainte d'horloges et λ un ensemble d'horloges. Ils existent des contraintes d'horloges ϕ_1 et ϕ_2 équivalentes à $\overrightarrow{\phi}$ et $\overleftarrow{\phi}^{\lambda}$, c'est-à-dire pour toute valuation d'horloges ν,

$\nu \models \overrightarrow{\phi}$ si et seulement si $\nu \models \phi_1$

$\nu \models \overleftarrow{\phi}^{\lambda}$ si et seulement si $\nu \models \phi_2$

Exemple :

Nous illustrons dans ce qui suit l'élimination d'une τ-transition sans remise à zéro d'horloges. Pour l'automate à entrées et sorties temporisé A, nous utilisons la clôture dans le

future de la contrainte d'horloges ϕ_2 afin d'éliminer la τ-transition et nous obtenons l'automate à entrées et sorties temporisé *B*. Il faut noter que la nouvelle horloge x_0 ainsi que la contrainte qui lui est associée sont nécessaires. Cette méthode ne peut pas être utilisée lorsqu'on désire éliminer une τ-transition qui n'est suivie par aucune autre transition. Afin d'éliminer une telle τ-transition dans l'automate à entrées et sorties temporisé *C*, nous utilisons la clôture par le passé de ϕ_2 par rapport à λ_1 et nous obtenons l'automate à entrées et sorties temporisé *D*.

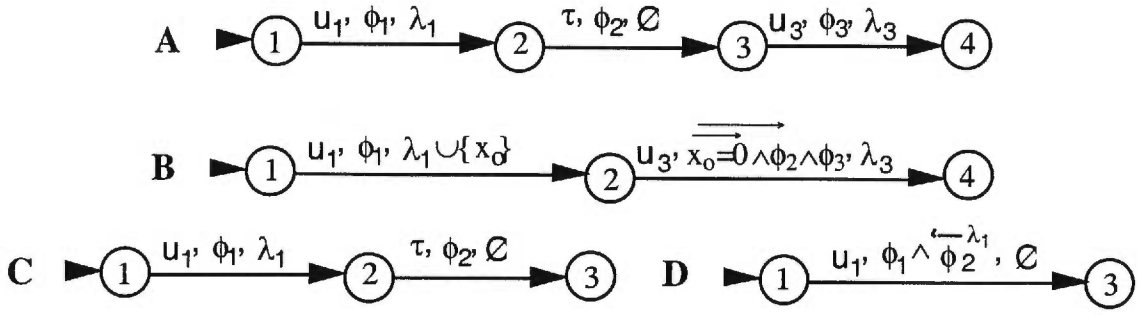


Figure 8.6 : Deux automates *A* et *C* et leurs équivalents sans τ-transitions *B* et *D*

8.3.2 Minimisation du nombre d'horloges d'un automate à entrées et sorties temporisé

Le nombre d'horloges utilisées dans une spécification croît principalement pour deux raisons. Premièrement, les spécifications sont souvent écrites dans des langages de description de haut niveau puis transformées par la suite en des automates temporisés ayant un nombre d'horloges proportionnel au nombre de contraintes temporelles qui apparaissent dans la spécification. Cependant, ces contraintes temporelles sont rarement actives au même moment, et donc le nombre d'horloges peut être réduit. Deuxièmement, les systèmes complexes sont décrits par la composition de composantes plus simples ayant chacune un petit nombre d'horloges. Cependant, à cause de la synchronisation de certaines transitions, plusieurs horloges sont remises à zéro simultanément et seront donc égale durant une certaine période puisque toutes les horloges avancent à la même vitesse que le temps. Il est claire dans ce cas, qu'une seule parmi ces horloges est réellement nécessaire.

En prenant en compte ces observation, les auteurs dans [Daws 96] proposent une méthode pour réduire le nombre d'horloges d'un automate temporisé en combinant deux algorithmes. Le premier permet de détecter les horloges actives. Intuitivement, une horloge est active dans un état si sa valeur dans l'état peut influencer l'évolution future du système.

Ceci peut arriver chaque fois que l'horloge apparaît dans l'invariant de l'état où dans la contrainte associée à une transition sortante. Les valeurs des horloges inactives à l'état *s* n'ont aucun impact sur l'évolution du système à partir de *s*. Ceci signifie que le nombre d'horloges nécessaires est égale au plus grand des nombres d'horloges actives dans chaque état. Le second permet de détecter les paires d'horloges qui sont toujours égales. Deux horloges sont égales dans un état, si dans toute transition entrante elles sont soit remise à zéro toutes les deux, soit elles sont égales dans l'état de départ de la transition et aucune des deux n'est remise à zéro.

8.3.3 Réduction des contraintes d'horloges

Étant donné un automate à entrées et sorties temporisé *A*, certaines transitions ne seront jamais exécutées puisque les contraintes d'horloges associées à ces transitions ont toujours la valeur fausse lorsqu'on atteint l'état correspondant. L'élimination de telles transitions ne change en rien l'ensemble des traces temporisées de *A*. Un certain nombre de travaux traitent ce problème [Courcoubetis 91][Somé 97]. Le graphe des régions [Alur 94] correspondant à l'automate à entrées et sorties temporisé *A* peut être utilisé pour l'élimination de telles transitions, mais la complexité est polynomiale en fonction du nombre d'états et de transitions de *A* et exponentielle en fonction du nombre d'horloges et de la longueur de la représentation binaire des constantes qui apparaissent dans les contraintes d'horloges. Dans [Somé 97], l'auteur propose un algorithme basé sur la propagation des relations d'horloges qui permet la détection de telles transitions. Intuitivement, pour chaque état nous déterminons l'ensemble des valuation d'horloges qui sont satisfaites dans l'état, puis pour chaque transition sortante nous considérons la restriction sur cet ensemble de la contrainte d'horloges associée. Toute transition dont la nouvelle contrainte a la valeur fausse sera éliminée.

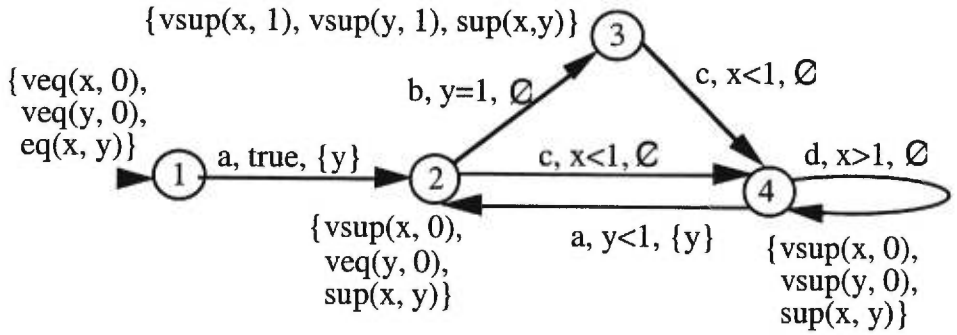


Figure 8.7 : propagation des relations d'horloges

Exemple :

Nous illustrons dans cet exemple la propagation des relations d'horloges. Pour l'exemple de la figure 8.7, dans l'état 3 vsup(x, 1) signifie que la valeur de l'horloge x est toujours supérieur à 1 dans cet état. La transition (3, c, x<1, Ø, 4) ne sera donc jamais exécutée, elle peut être éliminée.

8.4 La conception de sous-module

8.4.1 L'architecture

Nous utilisons la même architecture que celle illustrée dans la figure 7.1 du chapitre 7 afin de discuter le problème de la construction de la spécification d'un module inconnu d'un système composé. Nous considérons la classe de systèmes qui peuvent être modélisés par deux automates à entrées et sorties temporisés qui communiquent entre eux et avec l'environnement. Un automate à entrées et sorties temporisé, appelé le contexte C, modélise la partie connue du système, c'est-à-dire, celle qui existe au départ. Le module à concevoir peut observer un sous ensemble de l'ensemble des interactions entre l'environnement et le contexte C. Lorsque ce sous ensemble est égale à l'ensemble des interactions entre l'environnement et le contexte, nous avons le cas de l'observation totale [Maler 95]. Dans le cas contraire, nous sommes dans le cas de l'observation partielle. L'observation est minimale lorsque ce sous ensemble est vide.

8.4.2 Le problème

Dans ce chapitre, nous nous intéressons à la résolution de l'équation $(C||X)\leq_{\S} A$ sous la contrainte $I_X=In$ dans le modèle des automates à entrées et sorties temporisés. L'automate à entrées et sorties temporisé A, modélise le système complet désiré. L'ensemble In donné vérifiant $(I_A \backslash I_C)\cup(O_C \backslash O_A)\subseteq In\subseteq I_A \cup O_C$ décrit l'ensemble des entrées du module à concevoir X.

Nous proposons dans ce chapitre un algorithme et nous prouvons qu'il est correct. Cet algorithme permet de générer la solution générique de l'équation $(C||X)\leq_{\S} A$ sous la contrainte $I_X=In$, notée $Sol_{\S}$. De plus, nous caractérisons l'ensemble des solutions possibles de l'équation.

8.5 La solution pour la relation de conformité réalisation sécuritaire

8.5.1 L'approche proposée

Nous utilisons un automate à entrées et sorties temporisé chaotique qui représente toutes les traces sur l'alphabet d'entrées *In* et sur l'alphabet de sorties $(I_C \setminus I_A) \cup (O_A \setminus O_C)$. Toute solution de l'équation $(C \| X) \leq_{\mathfrak{S}} A$ a un ensemble de traces inclus dans l'ensemble des traces de l'automate chaotique. L'idée principale de la méthode consiste en l'élimination de toute trace contenue dans l'automate temporisé chaotique, dont la combinaison avec des traces temporisées du contexte C dans l'environnement associé par réflexion à la spécification du système A décrit au moins un comportement non conforme. Ceci permettra de capturer l'ensemble des traces temporisées permises (s'il n'est pas vide) de la composante à concevoir sous la forme d'un automate à entrées et sorties temporisé $Sol_{\mathfrak{S}}$, appelé la solution sécuritaire générique. Une trace temporisée permise est une trace temporisée d'un automate à entrées et sorties temporisé qui est une solution de l'équation $(C \| X) \leq_{\mathfrak{S}} A$ sous la contrainte $I_X = In$.

Pour réaliser ce but, nous construisons les formes complétées des automates à entrées et sorties temporisés de A' et C', où A' est obtenu à partir de A après réduction des contraintes temporelles et élimination des invariants, et C' est obtenu à partir de C après réduction des contraintes temporelles. Nous construisons par la suite la composition $R = Tief(C') \| Chaos \| Tief(\widetilde{A'})$ qui est au faite égale à $Tief(C') \| Tief(\widetilde{A'})$. Puisque nous considérons la réalisation sécuritaire comme relation de conformité, nous remplaçons tous les invariants présents dans R par la contrainte triviale *"true"*. Dans le cas d'observation partielle, c'est à dire, $In \subset I_A \cup O_C$, nous devons remplacer toutes les actions dont les étiquettes ne font pas partie de l'alphabet de la composante à concevoir par l'action interne τ. Nous supposons que les τ-transitions sont sans remise à zéro d'horloges. Par la suite, nous éliminons ces τ-transitions. L'automate obtenu sera en général non déterministe. Il est en général impossible de le déterminiser car la classe des automates à entrées et sorties temporisés déterministes est strictement incluse dans la classe des automates à entrées et sorties temporisés non déterministes [Alur 94a]. Nous devons alors transformer notre automate avant l'élimination des transitions qui mènent à l'état spécial *Fail*. Le but de cette transformation est d'obtenir un automate dans lequel deux états accessibles grâce à une trace temporisée commune acceptent le même ensemble d'entrées et produisent le même ensemble de sorties.

Nous proposons dans ce qui suit, un algorithme pour la construction de la solution générique sécuritaire.

8.5.2 L'algorithme

Notations :

v_0 : est la valuation d'horloge définie par $v_0(x)=0$ pour toute horloge x dans X_A;

φ^{λ} : est la contrainte dérivée à partir de φ en remplaçant les horloges dans λ par 0;

$CONSTR(\lambda)$: est l'ensemble contenant les contraintes $x=0$ pour toute horloge x dans λ;

Algorithme 8.1

Entrées : La spécification du contexte C, la spécification du système A et un ensemble d'entrées pour la solution *In*.

Sortie : un automate à entrées et sorties temporisé $Sol_{\S}$ tel que $I_{Sol_{\S}}=In$ qui satisfait l'équation $(C||Sol_{\S})\leq_{\S}A$ lorsqu'une solution existe.

Étape 1 : Compléter et réduire les contraintes temporelles des automates

Dans cette étape, nous construisons les formes complétées des automates à entrées et sorties temporisés A' et C', où A' est l'automate à entrées et sorties temporisé obtenu à partir de A après réduction des contraintes temporelles et élimination des invariants, et C' est l'automate à entrées et sorties temporisé obtenu à partir de C après réduction des contraintes temporelles.

Étape 2 : Composition

Dans cette étape, nous construisons l'automate composé $R:=Tief(C')||Tief(\widetilde{A'})$. Puisque un état de R est une paire contenant un état de chacun des automates $Tief(C')$ et $Tief(\widetilde{A'})$, nous remplacerons cet état par $Fail_R$ chaque fois qu'il contient $Fail_{C'}$ ou $Fail_{\widetilde{IA'}}$.

Étape 3 : Minimisation du nombre d'horloges

Dans cette étape, nous remplaçons tous les invariants présents dans R par "*true*" puis nous minimisons le nombre d'horloges dans R.

Étape 4 : Élimination des transitions invisibles

Dans cette étape, nous remplaçons dans R toutes les actions n'appartenant pas à $In\cup(I_C\setminus I_A)\cup(O_A\setminus O_C)$ par l'action invisible τ, puis nous transformons R en un automate à entrées et sorties temporisé équivalent ne contenant aucune transition invisible. Cette transformation est réalisée de la façon suivante, nous éliminons les transitions invisibles qui mènent à un état silencieux, puis pour chaque état p où des transitions invisibles sont présentes nous ajoutons une nouvelle horloge x_p qui sera remise à zéro par toute transition menant à l'état p. Par la suite,

• pour tout chemin $p_1 \xrightarrow{\varepsilon,\ \emptyset,\ \varphi_1} p_2 \ldots p_k \xrightarrow{\varepsilon,\ \emptyset,\ \varphi_k} p_{k+1} \xrightarrow{u,\ \lambda,\ \varphi_{k+1}} p_{k+2}$, où $(p_i)_{1\le i\le k+1}$ sont des états distincts, nous ajoutons la transition $(p_1, u, \lambda, \overrightarrow{x_{p_1}=0} \wedge \varphi_1 \wedge \ldots \wedge \varphi_k \wedge \varphi_{k+1}, p_{k+2})$.

• pour tout chemin $p_1 \xrightarrow{u,\ \lambda,\ \varphi_1} p_2 \xrightarrow{\varepsilon,\ \emptyset,\ \varphi_2} \ldots p_k \xrightarrow{\varepsilon,\ \emptyset,\ \varphi_k} Fail_{R_1}$, où $(p_i)_{1\le i\le k}$ sont des états distincts, nous ajoutons la transition $(p_1, u, \emptyset, \overleftarrow{\lambda}\ \varphi_1 \wedge \overrightarrow{x_{p_2}=0} \wedge \varphi_2 \wedge \ldots \wedge \varphi_{k-1} \wedge \varphi_k\ , Fail_{R_1})$.

• pour tout chemin $s_{oR} \xrightarrow{\varepsilon,\ \emptyset,\ \varphi_1} p_2 \ldots p_k \xrightarrow{\varepsilon,\ \emptyset,\ \varphi_k} Fail_R$, nous ajoutons la transition $(s_{oR}, \mathrm{E}, \emptyset, \overrightarrow{\varphi_1 \wedge \ldots \wedge \varphi_{k-1} \wedge \varphi_k}, Fail_{R_1})$ où E est un nouvelle étiquette.

Finalement, nous éliminons toutes les transitions invisibles. L'automate obtenu est noté R_1.

Étape 5 : Élimination des transitions menant à l'état *Fail*

Dans cette étape, nous éliminons de l'automate R_1 toutes les traces menant à $Fail_{R_1}$. Contrairement au cas non temporisé, une transition menant à l'état $Fail_{R_1}$ peut être évitée si il est possible de placer un invariant dans l'état de départ de la transition qui force à quitter l'état avant le temps d'occurrence de cette transition. Cette étape est réalisée en trois sous-étapes comme suit :

Sous-étape 5.1 : Dans cette partie, nous utilisons la fonction *Éliminer_*E*_transitions* qui reçoit en entrée un automate à entrées et sorties temporisé et retourne si possible un automate à entrées et sorties temporisé ne contenant aucune E*_transitions* sinon elle retourne "PAS DE SOLUTION". Une E*_transition* est due à une suite d'interactions entre l'environnement et le contexte menant à l'état *Fail* sans intervention de la composante à concevoir. La fonction *Éliminer_*E*_transitions* détermine un invariant à placer dans l'état initial de la composante à concevoir de manière que l'interaction précoce de cette dernière avec l'environnement ou le contexte empêche que la suite d'interactions entre l'environnement et le contexte menant à l'état *Fail* soit complétée.

Éliminer_*E*_transitions*(*TIOA* R_1) : *TIOA

TANT QUE il existe une transition $(s_{oR}, \mathrm{E}, \emptyset, \varphi_1, Fail_{R_1})$ **FAIRE**

CONSTRANS=$\{\varphi \mid (s_{oR}, u, \lambda, \varphi, s') \in T_{R_1}, s' \neq Fail_{R_1}$ et $u \in (I_C \setminus I_A) \cup (O_A \setminus O_C)\}$;

$\Phi = (\bigvee_{\varphi \in CONSTRANS} \varphi) \wedge (\overline{\varphi_1}^{\emptyset} \wedge \neg \varphi_1)$;

SI ($v_0 \models \overline{\varphi_1}^{\emptyset}$) **ALORS**

SI $\Phi \wedge \overrightarrow{CONSTR(X_{R_1})} \neq \emptyset$ **ALORS**

Éliminer la transition (s_{0R}, E, $\emptyset$, φ_1, $Fail_{R_1}$);

$Inv(s_{0R}) := Inv(s_{0R}) \wedge (\overline{\Phi}^{\emptyset} \vee \neg \overline{\varphi_1}^{\emptyset})$;

Réduire toutes les contraintes dans R_1;

POUR chaque transition (s, u, λ, φ, s_{0R})$\in T_{R_1}$ **FAIRE**

SI $\varphi^{\lambda} \wedge \neg Inv(s_{0R}) \neq \emptyset$ **ALORS**

Remplacer (s, u, λ, φ, s_{0R}) par (s, u, λ, $\varphi \wedge Inv(s_{0R})^{\lambda}$, s_{0R});

Ajouter dans R_1 la transition (s, u, $\emptyset$, $\varphi \wedge \neg Inv(s_{0R})^{\lambda}$, $Fail_{R_1}$);

SINON retourner " PAS DE SOLUTION"; **STOP**;

SINON

Éliminer la transition (s_{0R}, E, $\emptyset$, φ_1, $Fail_{R_1}$);

SI $\Phi \neq \emptyset$ **ALORS**

$Inv(s_{0R}) := Inv(s_{0R}) \wedge (\overline{\Phi}^{\emptyset} \vee \neg \overline{\varphi_1}^{\emptyset})$;

Réduire toutes les contraintes dans R_1;

POUR chaque transition (s, u, λ, φ, s_{0R})$\in T_{R_1}$ **FAIRE**

SI $\varphi^{\lambda} \wedge \neg Inv(s_{0R}) \neq \emptyset$ **ALORS**

Remplacer (s, u, λ, φ, s_{0R}) par (s, u, λ, $\varphi \wedge Inv(s_{0R})^{\lambda}$, s_{0R});

Ajouter dans R_1 la transition (s, u, $\emptyset$, $\varphi \wedge \neg Inv(s_{0R})^{\lambda}$, $Fail_{R_1}$);

SINON

POUR chaque transition (s, u, λ, φ, s_{0R})$\in T_{R_1}$ **FAIRE**

SI $\varphi^{\lambda} \wedge \overline{\varphi_1}^{\emptyset} \neq \emptyset$ **ALORS**

Remplacer (s, u, λ, φ, s_{0R}) par (s, u, λ, $\varphi \wedge (\neg \overline{\varphi_1}^{\emptyset})^{\lambda}$, s_{0R});

Ajouter dans R_1 la transition (s, u, $\emptyset$, $\varphi \wedge (\overline{\varphi_1}^{\emptyset})^{\lambda}$, $Fail_{R_1}$);

Sous-étape 5.2 : Puisque la composante à concevoir sera obtenue sous la forme d'un automate à entrées et sorties temporisé non-déterministe, l'environnement et le contexte ne peuvent observer de façon précise l'état de la composante. Tous les états de la composante qui sont accessibles par une trace temporisée commune doivent accepter le même ensemble d'entrées. Pour cela, nous utilisons la fonction *AugmentInputs* qui prendra comme argument l'automate à entrées et sorties temporisé obtenu à la fin de la sous-étape 5.1 et retournera un

automate à entrées et sorties temporisé ayant la propriété désirée.

AugmentInputs(TIOA R_1) : TIOA

POUR chaque état $s \neq Fail_{R_1}$ dans R_1 **FAIRE**

$TEMP = R_1$;

Éliminer à partir de *TEMP* tous les états à partir desquels on ne peut pas atteindre s;

$I_{TEMP} = I_{R_1} \cup O_{R_1}$;

$O_{TEMP} = \emptyset$;

Renommer toutes les horloges dans *TEMP*;

$TEMP = TEMP \,\|R_1$;

Éliminer à partir de *TEMP* les états $(s', Fail_{R_1})$ et toutes les transitions menant à eux;

Éliminer à partir de *TEMP* tous les états à partir desquels on ne peut pas atteindre un état de la forme (s, s') avec $s' \neq s$ et $s' \neq Fail_{R_1}$;

Ajouter à R_1 les états (s_1, s_2) de *TEMP* tels que $s_1 \neq s_2$;

POUR chaque état (s_1, s_2) de *TEMP* **FAIRE**

SI $s_1 \neq s_2$ **ALORS**

POUR chaque transition $((s_1, s_2), u, \lambda, \varphi, (s_3, s_4))$ dans *TEMP* **FAIRE**

SI $s_3 \neq s_4$ **ALORS**

Ajouter la transition $((s_1, s_2), u, \lambda, \varphi, (s_3, s_4))$ à R_1;

SINON

Ajouter la transition $((s_1, s_2), u, \lambda, \varphi, s_3)$ à R_1;

Ajouter la transition $((s_1, s_2), u, \lambda', \varphi', s_4)$ à R_1 pour compléter la transition correspondante $(s_2, u, \lambda', \varphi'', s_4)$ dans R_1 avec $\varphi' = \varphi'' \wedge \neg\varphi$;

POUR chaque transition $(s_2, u', \lambda', \varphi', s')$ dans R_1 **FAIRE**

SI u' n'est pas étiquette d'une transition sortante de (s_1, s_2) dans R_1

ALORS

Ajouter la transition $((s_1, s_2), u', \lambda', \varphi', s')$ dans R_1;

SINON

POUR chaque transition $((s_1, s_2), u, \lambda, \varphi, (s_3, s_4))$ dans *TEMP* **FAIRE**

SI $s_3 \neq s_4$ **ALORS**

Ajouter la transition $(s_1, u, \lambda, \varphi, (s_3, s_4))$ dans R_1;

Remplacer la transition $(s_1, u, \lambda', \varphi', s_4)$ dans R_1 par $(s_1, u, \lambda', \varphi' \wedge \neg\varphi, s_4)$;

POUR chaque état (s, s_2) de R_1 **FAIRE**

POUR chaque transition $(s, u, \lambda, \varphi, s_3)$ dans R_1 **FAIRE**

SI $u \in I_{R_1}$ **ALORS**

$PresConst := \{\varphi' \mid ((s, s_2), u, \lambda', \varphi', s')$ dans $R_1\}$;

$\varphi'' := \varphi \wedge \neg(\bigvee_{\varphi' \in PresConst} \varphi')$;

Ajouter la transition $((s, s_2), u, \lambda, \varphi'', s_3)$ dans R_1;

Sous-étape 5.3 : Maintenant nous sommes prêt à traiter les transitions menant à l'état *Fail*. Nous utilisons la fonction *Éliminer_Fail* qui reçoit en entrée l'automate à entrées et sorties temporisé obtenu à la fin de la sous-étape 5.2 et essaie d'éliminer lorsque cela est possible toutes les traces temporisés menant à l'état *Fail*. Pour chaque transition $(s, u, \emptyset, \varphi_2, Fail_{R_1})$, nous propageons d'abord la transition, c'est à dire, nous transformons l'automate afin d'obtenir un automate où tout état, accessible à partir de l'état initial par une trace temporisée menant à l'état s, mène avec l'action u et sous la contrainte φ_2 à l'état $Fail_{R_1}$. Pour éliminer la transition il y a plusieurs cas. Par exemple (Figure 8.2), si $u=x_1$ est une entrée et $Inv(s)=true$, nous considérons la contrainte $\Phi = (\bigvee_{\varphi \in CONSTRANS} \varphi) \wedge (\overleftarrow{\varphi_2}^{\emptyset} \wedge \neg \varphi_2)$ où *CONSTRANS* est l'ensemble de toutes les contraintes associées aux transitions sortantes de s et étiquetées par des sorties. Si $\Phi \neq$false, on pose $Inv(s)= \overleftarrow{\Phi}^{\emptyset} \vee \neg \overleftarrow{\varphi_2}^{\emptyset}$ afin d'empêcher la réalisation de la transition $(s, x_1, \emptyset, \varphi_2, Fail_{R_1})$ et chaque transition entrante dans s $(s', u', \lambda', \varphi', s)$ est remplacée par deux transitions $(s', u', \lambda', \varphi' \wedge Inv(s)^{\lambda'}, s)$ et $(s', u', \emptyset, \varphi' \wedge \neg Inv(s)^{\lambda'}, Fail_{R_1})$ où $Inv(s)^{\lambda'}$ est l'ensemble des contraintes dérivées à partir $Inv(s)$ en éliminant les contraintes reliées à des horloges dans λ', sinon nous éliminons toutes les transitions sortantes de s puis nous remplaçons s par $Fail_{R_1}$.

Si nous obtenons un automate à entrées et sorties temporisé à la fin de cette étape nous l'appelons Sol_5.

Éliminer_Fail(TIOA R_1) : TIOA

TANT QUE il existe une transition $(s, u, \lambda, \varphi_2, Fail_{R_1})$ dans R_1 **FAIRE**

$TEMP := R_1$;

Éliminer à partir de *TEMP* tous les états à partir desquels on ne peut pas atteindre s;

Ajouter à *TEMP* l'état $Fail_{TEMP}$ et la transition $(s, u, \lambda, \varphi_2, Fail_{TEMP})$;

$I_{TEMP}=I_{R_1} \cup O_{R_1}$; $O_{TEMP}=\emptyset$;

Renommer toutes les horloges dans *TEMP*;

$TEMP=TEMP \,\|R_1$;

Éliminer à partir de *TEMP* les états $(s', Fail_{R_1})$ et toutes les transitions menant à eux;

Éliminer à partir de *TEMP* tous les états à partir desquels on ne peut pas atteindre un

état sous la forme $(Fail_{TEMP}, s')$ avec $s' \neq Fail_{R_1}$;

Ajouter à R_1 les états (s_1, s_2) de *TEMP* tels que $s_1 \neq s_2$;

POUR chaque état (s_1, s_2) de *TEMP* tel que $s_1 \neq Fail_{TEMP}$ **FAIRE**

SI $s_1 \neq s_2$ **ALORS**

POUR chaque transition $((s_1, s_2), u_1, \lambda_1, \varphi_1, (s_3, s_4))$ dans *TEMP* **FAIRE**

SI $s_3 \neq s_4$ **ALORS** Ajouter la transition $((s_1, s_2), u_1, \lambda_1, \varphi_1, (s_3, s_4))$ dans R_1;

SINON Ajouter la transition $((s_1, s_2), u_1, \lambda_1, \varphi_1, s_3)$ dans R_1;

Ajouter la transition $((s_1, s_2), u_1, \lambda', \varphi', s_4)$ dans R_1 afin de compléter la transition correspondante $(s_2, u_1, \lambda', \varphi'', s_4)$ dans R_1 avec $\varphi' = \varphi'' \wedge \neg \varphi_1$;

POUR chaque transition $(s_2, u', \lambda', \varphi', s')$ dans R_1 **FAIRE**

SI u' n'est pas une étiquette d'une transition sortante de (s_1, s_2) **ALORS**

Ajouter la transition $((s_1, s_2), u', \lambda', \varphi', s')$ dans R_1;

SINON

POUR chaque transition $((s_1, s_2), u_1, \lambda_1, \varphi_1, (s_3, s_4))$ dans *TEMP* **FAIRE**

SI $s_3 \neq s_4$ **ALORS**

Ajouter la transition $(s_1, u_1, \lambda, \varphi, (s_3, s_4))$ dans R_1;

Remplacer la transition $(s_1, u, \lambda', \varphi', s_4)$ dans R_1 par $(s_1, u, \lambda', \varphi' \wedge \neg \varphi, s_4)$;

Remplacer chaque état $(Fail_{TEMP}, s')$ par $Fail_{R_1}$;

SI $s \neq s_{0R_1}$ **ALORS**

SI $u \in (I_C \backslash I_A) \cup (O_A \backslash O_C)$ **ALORS**

SI $\neg Inv(s) \wedge \overrightarrow{\varphi_2} = \emptyset$ **ALORS**

Éliminer la transition $(s, u, \lambda, \varphi_2, Fail_{R_1})$ de R_1;

Remplacer dans R_1 chaque transition $(s, u, \lambda', \varphi', s')$ par $(s, u, \lambda', \neg \varphi_2 \wedge \varphi', s')$;

SINON

CONSTRANS$= \{ \varphi \mid (s, u, \lambda, \varphi, s') \in T_{R_1}, s' \neq Fail_{R_1}$ et $u \in (I_C \backslash I_A) \cup (O_A \backslash O_C) \}$;

SI $Inv(s) \wedge \neg \left(\overleftrightarrow{\varphi_2} \wedge \neg \left(\bigvee_{\varphi \in CONSTRANS} \overrightarrow{\varphi} \right) \neg Inv(s) \right) = \emptyset$ **ALORS**

Éliminer toutes les transitions sortante de s;

Remplacer s par $Fail_{R_1}$;

$R_1 := CC(R_1)$;

SINON

Éliminer la transition $(s, u, \lambda, \varphi_2, Fail_{R_1})$ de R_1;

Remplacer dans R_1 chaque transition $(s, u, \lambda', \varphi', s')$ par $(s, u, \lambda',$

$\neg\varphi_2\wedge\varphi', s'$);

$Inv(s)=Inv(s)\wedge\neg(\overleftarrow{\overrightarrow{\varphi_2}}\wedge\neg(\bigvee_{\varphi\in CONSTRANS}\overleftarrow{\overrightarrow{\varphi}})\neg Inv(s))$;

Réduire toutes les contraintes dans R_1;

POUR chaque transition $(s', u', \lambda', \varphi', s)\in T_{R_1}$ **FAIRE**

SI $CONSTR(\lambda')\wedge Inv(s)\neq\emptyset$ **ALORS**

SI $(\varphi')^{\lambda'}\wedge\neg Inv(s)\neq\emptyset$ **ALORS**

Remplacer $(s', u', \lambda', \varphi', s)$ par $(s', u', \lambda', \varphi'\wedge Inv(s)^{\lambda'}, s)$;

Ajouter dans R_1 la transition $(s', u', \emptyset, \varphi'\wedge\neg Inv(s)^{\lambda'}, Fail_{R_1})$;

SINON Remplacer $(s', u', \lambda', \varphi', s)$ par $(s', u', \emptyset, \varphi', Fail_{R_1})$;

SINON

CONSTRANS=$\{\varphi \mid (s, u, \lambda, \varphi, s')\in T_{R_1}, s'\neq Fail_{R_1}$ et $u\in(I_C\backslash I_A)\cup(O_A\backslash O_C)\}$;

$\Phi=(\bigvee_{\varphi\in CONSTRANS}\varphi)\wedge(\overleftarrow{\varphi_2}\wedge\neg\varphi_2)$;

SI $Inv(s)\wedge(\overleftarrow{\Phi}\vee\neg\overleftarrow{\varphi_2})=\emptyset$ **ALORS**

Éliminer toutes les transitions sortante de s;

Remplacer s par $Fail_{R_1}$;

$R_1:=CC(R_1)$;

SINON

Éliminer la transition $(s, u, \lambda, \varphi_2, Fail_{R_1})$;

$Inv(s):=Inv(s)\wedge(\overleftarrow{\Phi}\vee\neg\overleftarrow{\varphi_2})$;

Réduire toutes les contraintes dans R_1;

POUR chaque transition $(s', u', \lambda', \varphi', s)\in T_{R_1}$ **FAIRE**

SI $CONSTR(\lambda')\wedge Inv(s)\neq\emptyset$ **ALORS**

SI $(\varphi')^{\lambda'}\wedge\neg Inv(s)\neq\emptyset$ **ALORS**

Remplacer $(s', u', \lambda', \varphi', s)$ par $(s', u', \lambda', \varphi'\wedge Inv(s)^{\lambda'}, s)$;

Ajouter dans R_1 la transition $(s', u', \emptyset, \varphi'\wedge\neg Inv(s)^{\lambda'}, Fail_{R_1})$;

SINON Remplacer $(s', u', \lambda', \varphi', s)$ par $(s', u', \emptyset, \varphi', Fail_{R_1})$;

SINON

SI $u\in(I_C\backslash I_A)\cup(O_A\backslash O_C)$ **ALORS**

SI $\neg Inv(s)\wedge\overrightarrow{\varphi_2}=\emptyset$ **ALORS**

Éliminer la transition $(s, u, \lambda, \varphi_2, Fail_{R_1})$;

Remplacer chaque transition $(s, u, \lambda', \varphi', s')$ par $(s, u, \lambda', \neg\varphi_2\wedge\varphi', s')$;

SINON

CONSTRANS=$\{\varphi \mid (s, u, \lambda, \varphi, s') \in T_{R_1}, s' \neq Fail_{R_1}$ et $u \in (I_C \setminus I_A) \cup (O_A \setminus O_C)\}$;

SI $(v_0 \models Inv(s)) \wedge \neg(\overleftarrow{\varphi_2} \wedge \neg(\bigvee_{\varphi \in CONSTRANS} \overrightarrow{\varphi}) \neg Inv(s))$ **ALORS**

Éliminer la transition $(s, u, \lambda, \varphi_2, Fail_{R_1})$;

Remplacer chaque transition $(s, u, \lambda', \varphi', s')$ par $(s, u, \lambda', \neg\varphi_2 \wedge \varphi', s')$;

$Inv(s) = Inv(s) \wedge \neg(\overleftarrow{\varphi_2} \wedge \neg(\bigvee_{\varphi \in CONSTRANS} \overrightarrow{\varphi}) \neg Inv(s))$;

Réduire toutes les contraintes dans R_1;

POUR chaque transition $(s', u', \lambda', \varphi', s) \in T_{R_1}$ **FAIRE**

SI $(\varphi')^{\lambda'} \wedge \neg Inv(s) \neq \emptyset$ **ALORS**

Remplacer $(s', u', \lambda', \varphi', s)$ par $(s', u', \lambda', \varphi' \wedge Inv(s)^{\lambda'}, s)$;

Ajouter dans R_1 la transition $(s', u', \emptyset, \varphi' \wedge \neg Inv(s)^{\lambda'}, Fail_{R_1})$;

SINON retourner " PAS DE SOLUTION"; **STOP**;

SINON

CONSTRANS=$\{\varphi \mid (s, u, \lambda, \varphi, s') \in T_{R_1}, s' \neq Fail_{R_1}$ et $u \in (I_C \setminus I_A) \cup (O_A \setminus O_C)\}$;

$\Phi = (\bigvee_{\varphi \in CONSTRANS} \varphi) \wedge (\overleftarrow{\varphi_2}^{\emptyset} \wedge \neg \varphi_2)$;

SI $(v_0 \models Inv(s)) \wedge (\overleftarrow{\Phi}^{\emptyset} \vee \neg \overleftarrow{\varphi_2}^{\emptyset})$ **ALORS**

Éliminer la transition $(s, u, \lambda, \varphi_2, Fail_{R_1})$;

$Inv(s) := Inv(s) \wedge (\overleftarrow{\Phi}^{\emptyset} \vee \neg \overleftarrow{\varphi_2}^{\emptyset})$;

Réduire toutes les contraintes dans R_1;

POUR chaque transition $(s', u', \lambda', \varphi', s) \in T_{R_1}$ **FAIRE**

SI $(\varphi')^{\lambda'} \wedge \neg Inv(s) \neq \emptyset$ **ALORS**

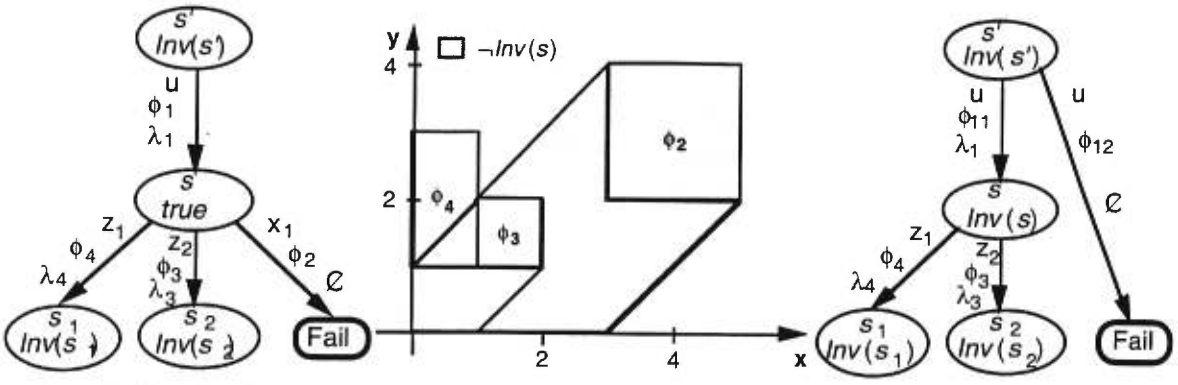
Remplacer $(s', u', \lambda', \varphi', s)$ par $(s', u', \lambda', \varphi' \wedge Inv(s)^{\lambda'}, s)$;

Ajouter dans R_1 la transition $(s', u', \emptyset, \varphi' \wedge \neg Inv(s)^{\lambda'}, Fail_{R_1})$;

SINON retourner " PAS DE SOLUTION"; **STOP**;

8.5.3 Exemple illustrant l'élimination d'une transition dans l'Étape 5

Dans l'exemple suivant, nous illustrons la technique d'élimination d'une transition menant à $Fail_{R_1}$. Nous considérons l'état s avec ses transitions entrantes et sortantes (Figure 8.8 avant l'élimination). La transition que nous allons éliminer est $(s, x_1, \emptyset, \varphi_2, Fail_{R_1})$. Pour cela, nous avons :



Avant l'élimination

Après l'élimination

Figure 8.8 : Élimination d'une transition menant à l'état $Fail_{R_1}$

la contrainte φ_1 est : $(x\leq5)\wedge(y\leq4)\wedge(y\geq2)$ et $\lambda_1=\{y\}$,
la contrainte φ_2 est : $(x\leq5)\wedge(x\geq3)\wedge(y\leq4)\wedge(y\geq2)$,
la contrainte φ_3 est : $(x\leq2)\wedge(x\geq1)\wedge(y\leq2)\wedge(y\geq1)$,
et la contrainte φ_4 est : $(x\leq1)\wedge(y\leq3)\wedge(y\geq1)$.
La contrainte Φ calculée dans la sous-étape 5.3 est : $(x\leq2)\wedge(y\leq2)\wedge(y-x\leq1)$,
Puisque $\Phi\neq$false, la contrainte $\overleftarrow{\Phi}^{\emptyset}$ est : $(x\leq2)\wedge(y\leq2)\wedge(y-x\leq1)\wedge(x-y\leq1)$, et la contrainte $\overleftarrow{\varphi_2}^{\emptyset}$ est : $(y-x\leq1)\wedge(y\leq4)\wedge(x\leq5)\wedge(x-y\leq3)$.
Donc $Inv(s)$ sera : $((x\leq2)\wedge(y\leq2)\wedge(y-x\leq1)\wedge(x-y\leq1))\vee(y-x>1)\vee(y>4)\vee(x>5)\vee(x-y>3)$,
et la contrainte $Inv(s)^{\{y\}}$ est : $(x\leq1)\vee(x>3)$,
Finalement, φ_{11} sera : $((x\leq1)\vee(x>3))\wedge(y\leq4)\wedge(y\geq2)$ et φ_{12} sera : $(x>1)\wedge(x\leq3)\wedge(y\leq4)\wedge(y\geq2)$.

8.5.4 L'ensemble des solutions

Théorème 8.1 : Étant donnés deux automates à entrées et sorties temporisés A et C, et étant donné un ensemble In tel que $(I_A\backslash I_C)\cup(O_C\backslash O_A)\subseteq In\subseteq I_A\cup O_C$, si l'algorithme 8.1 produit un automate à entrées et sorties temporisé $Sol_{\mathfrak{S}}$ alors c'est une solution au problème de construction de sous modules, c'est à dire, $(C \parallel Sol_{\mathfrak{S}}) \leq_{\mathfrak{S}} A$ et $I_X=In$, sinon il n'existe aucune solution pour ce problème ayant In comme ensemble d'entrées.

La solution obtenue par la méthode décrite précédemment est générique. On peut dériver à partir d'elle l'ensemble des solutions pour l'équation $(C \parallel X) \leq_{\mathfrak{S}} A$ sous la contrainte $I_X=In$.

Théorème 8.2 : Étant donnés deux automates à entrées et sorties temporisés A et C, et

étant donné un ensemble In tel que $(I_A \backslash I_C) \cup (O_C \backslash O_A) \subseteq In \subseteq I_A \cup O_C$, si l'algorithme 8.1 produit un automate à entrées et sorties temporisé $Sol_{\mathbb{S}}$ alors pour tout automate à entrées et sorties temporisé B, avec $I_B = In$ et $O_B = (I_C \backslash I_A) \cup (O_A \backslash O_C)$, les propositions suivantes sont équivalentes :

i - $B \leq_{\mathbb{TS}} Sol_{\mathbb{S}}$,

ii - B est une solution de l'équation $(C \parallel X) \leq_{\mathbb{S}} A$.

Les preuves des théorèmes 8.1 et 8.2 sont semblables à celles des théorèmes 7.1 et 7.2 présentées dans le chapitre 7.

CHAPITRE 9

L'OUTIL SCT POUR LA CONSTRUCTION DE SOUS-MODULES

Une continuation naturelle du travail décrit dans les chapitres précédant consiste à développer un outil pour la construction de sous-modules. Le langage de programmation orienté-objet Java a été choisi pour l'implantation des algorithmes décrits dans les chapitres 5, 6 et 7. Le choix du langage Java réside dans ses qualités de portabilité et de rapidité dans le développement. Le but principal de l'outil est la génération de la solution générique pour le problème de la construction de sous-modules dans le cas où les spécifications sont données sous la forme de machines de Mealy ou bien sous la forme d'automates à entrées et sorties et ceci pour une variété de relations de conformité. Une solution générique est une solution à partir de laquelle on peut dériver toutes les solutions possibles.

9.1 Description de l'outil pour la construction de sous-modules

9.1.1 Les possibilités de l'outil

Pour le modèle des machines de Mealy, l'outil pour la construction de sous-modules offre les possibilités suivantes:

(1) la construction de la solution générique dans le cas où les machines de Mealy décrivant le contexte et le système entier sont :

- déterministes et complètement spécifiées et où la relation de conformité utilisée est l'équivalence de traces,
- non déterministes et complètement spécifiées et où la relation de conformité utilisée est la réduction,
- partiellement spécifiées et où la relation de conformité utilisée est la quasi-équivalence,

(2) la construction de la solution minimale en prenant comme critère d'optimisation le nombre d'états,

(3) la vérification qu'une machine de Mealy donnée est une solution pour une équation donnée,

(4) la construction de la réduction déterministe minimale d'une machine de Mealy non déterministe observable donnée en prenant comme critère d'optimisation le nombre d'états.

Pour le modèle des automates à entrées et sorties, l'outil offre les possibilités suivantes :

(1) la construction de la solution générique dans le cas où la relation de conformité utilisée est la réalisation sécuritaire,

(2) la construction de la solution générique dans le cas où la spécification du contexte est donnée sous la forme d'un automate à entrées et sorties, la spécification du système entier est donnée sous la forme d'un automate à entrées et sorties avec traces complètes optionnelles et où la relation de conformité utilisée est l'implantation conforme,

(3) la vérification qu'un automate à entrées et sorties donné est une réalisation sécuritaire d'un automate à entrées et sorties donné,

(4) la vérification qu'un automate à entrées et sorties donné est implantation conforme d'un automate à entrées et sorties avec traces complètes optionnelles donné A,

(5) la construction de la composition de deux automates à entrées et sorties donnés,

(6) la construction d'un automate à entrées et sorties déterministe à partir d'un automate à entrées et sorties où certaines actions sont rendues invisibles,

(7) la construction à partir d'un automate à entrées et sorties déterministe d'un automate à entrées et sorties déterministe ayant le même ensemble de traces et le nombre minimal d'états.

9.1.2 Description de l'implantation de l'outil

Lors de l'implantation de l'outil dans le langage de programmation orienté-objet Java, nous avons utilisé un certain nombre de classes. Nous donnons dans ce qui suit la description des principales classes. De plus amples informations ainsi qu'une copie du code de l'outil sont disponibles à l'adresse URL suivante : http://www.iro.umontreal.ca/~drissi.

- La classe Menu : cette classe réalise l'interface usager de l'outil.
- La classe IOAutomaton : elle s'occupe de la représentation interne d'un automate à entrées et sorties et offre des méthodes telles que la composition d'automates, le

masquage de certaines actions ainsi que la minimisation.

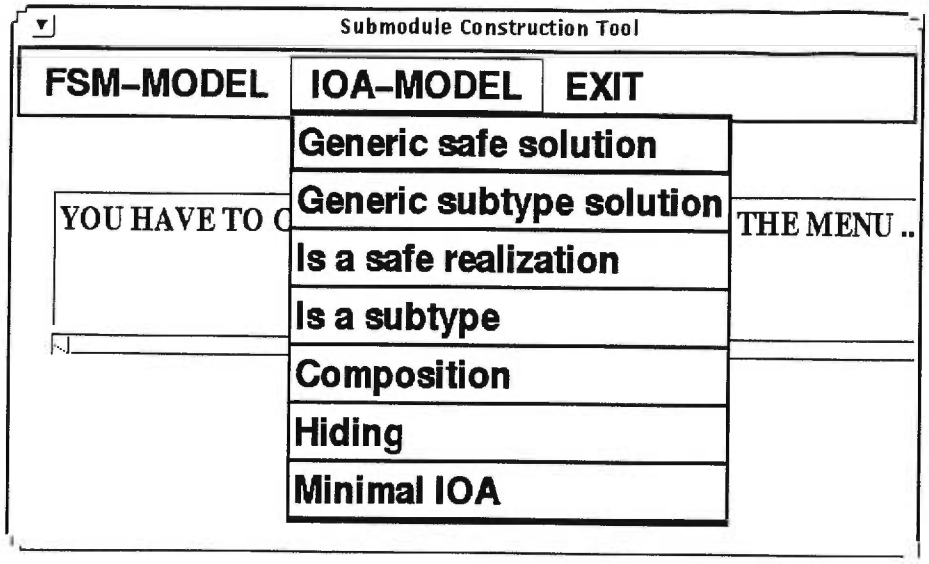


Figure 9.1 : Une vue de l'interface de l'outil

- La classe IOAWOCT : elle s'occupe de la representation interne d'un automate à entrées et sorties avec traces complètes optionnelles et offre les méthodes associées telles que la composition d'automates, le masquage de certaines actions ainsi l'élimination de traces interdites.
- La classe ReadFile : cette classe permet de transformer la forme textuelle de la spécification reçu en entrée sous la forme de la représentation interne.
- La classe OutFile : cette classe permet la transformation inverse. Elle produit à partir de la représentation interne des résultats une forme textuelle dirigée vers la sortie.
- La classe SafeR : Cette classe implante les étapes de l'algorithme pour la construction de la solution générique pour la relation de conformité réalisation sécuritaire.
- La classe Subtype : Cette classe implante les étapes de l'algorithme pour la construction de la solution générique pour la relation de conformité implantation conforme.
- La classe IsSafeR : Cette classe implante l'algorithme qui permet de vérifier si un automate à entrées et sorties donné est solution d'une équation donné lorsque la relation de conformité utilisée est la réalisation sécuritaire.
- La classe IsSubtype : Cette classe implante l'algorithme qui permet de vérifier si un automate à entrées et sorties donné est solution d'une équation donné lorsque la relation de conformité utilisée est la relation implantation conforme.

9.2. Illustration à travers un exemple du fonctionnement de l'outil

Dans ce qui suit, nous illustrons le fonctionnement de l'outil grâce à un exemple. Nous considérons le modèle des automates à entrées et sorties. Nous nous intéressons à la construction de la solution générique de l'équation $(C \| X) \leq_{conf} A$ sous la contrainte $I_X = In$. Pour la spécification du contexte C, nous associons l'automate à entrées et sorties de la Figure 9.2 ayant pour ensemble d'entrées $I_C = \{x_1, x_2, z_1, z_2, z_3, z_4\}$ et pour ensemble de sorties $O_C = \{u, y_1, y_2\}$.

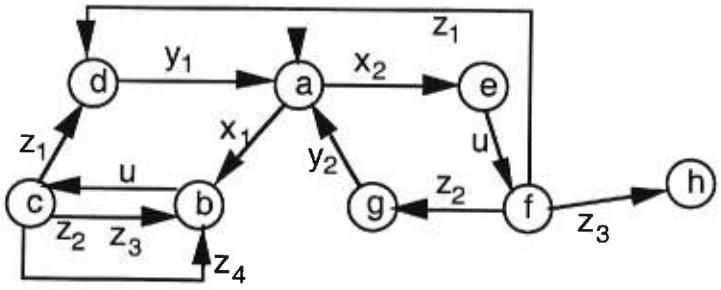


Figure 9.2 : L'automate à entrées et sorties C

Pour la spécification du système complet désiré, nous associons l'automate à entrées et sorties avec traces complètes optionnelles $A = (IOA_A, MT_A, OCT_A)$. L'automate à entrées et sorties IOA_A est celui décrit dans la Figure 9.3, il a pour ensemble d'entrées $I_{IOA_A} = \{x_1, x_2, x_3\}$ et pour ensemble de sorties $O_{IOA_A} = \{y_1, y_2, y_3\}$.

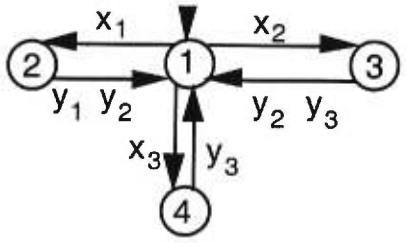


Figure 9.3 : L'automate à entrées et sorties IOA_A

L'ensemble décrivant les contraintes sur les traces obligatoire est :

$$MT_A = \{(1, \emptyset), (2, \{\{y_1, y_2\}\}), (3, \{\{y_2\}\}), (4, \{\{y_3\}\})\},$$

dans ce cas un élément $(s, \{Y\})$ dans MT_A impose qu'au moins un élément dans Y est une sortie obligatoite dans l'état s. L'élément $(2, \{y_1, y_2\}$ dans MT_A par exemple, impose que le système composé du contexte et du module à concevoir après la reception de l'entrée x_1 doit

toujours produire un sortie soit y_1 soit y_2. Nous remarquons que dans ce cas la relation implantation conforme représente la relation réduction.

L'ensemble décrivant les contraintes sur les traces complètes optionnelles est :

$$OCT_A=\{(1, \emptyset), (2, \{y_1, y_2\}), (3, \{y_2\}), (D, \{y_3\})\}.$$

L'ensemble des entrées requis pour le module à concevoir est $In=\{x_1, x_3, u\}$. Ceci entraine que le module à concevoir observera l'interaction entre le contexte et l'environnement portant l'étiquette x_1, mais n'observera pas les interactions entre le contexte et l'environnement portant les étiquettes x_2, y_1 et y_2. L'ensemble des sorties du module à concevoir sera dans ce cas $\{z_1, z_2, z_3, z_4, y_3\}$.

Après le démarrage de l'outil, on devra dans un premier temps choisir dans le menu déroulant correspondant au modèle des automate à entrées et sorties l'opération désirée. Dans notre cas nous choisirons l'opération :

Generic subtype solution

Dans ce cas l'outil demandera de rentrer dans des champs de saisie de texte les nom de trois fichier. Le premier fichier devra contenir la spécification du contexte. Le deuxième fichier devra contenir la spécification du système complet désiré. Enfin, le troisième fichier devra contenir l'ensemble des entrées requis pour le module à concevoir. La Figure 9.4 illustre la syntaxe du fichier associé à l'ensemble des entrées. Par contre la Figure 9.5 illustre la syntaxe des fichiers associés aux spécifications du contexte et du système complet désiré.

```
// La spécification de l'ensembles des entrées
x1;
x3;
u;
END;
```

Figure 9.4 : La syntaxe du fichier contenant les entrées requises

```
//La spécification du contexte
// Ensemble des entrées
x1;
x2;
z1;
z2;
z3;
z4;
END;
//Ensemble des sorties
u;
y1;
y2;
END;
//Ensemble des états
a;
b;
c;
d;
e;
f;
g;
END;
// Ensemble des transitions
a - x1 - b;
b - u - c;
c - z1 - d;
c - z2 - b;
c - z3 - b;
c - z4 - b;
d - y1 - a;
a - x2 - e;
e - u - f;
f - z1 - d;
f - z2 - g;
f - z3 - h;
g - y2 - a;
END;
```

```
//La spécification du système
// Ensemble des entrées
x1;
x2;
x3;
END;
//Ensemble des sorties
y1;
y2;
y3;
END;
//Ensemble des états
1;
2;
3;
4;
END;
// Ensemble des transitions
1 - x1 - 2;
1 - x2 - 3;
1 - x3 - 4;
2 - y1 - 1;
2 - y2 - 1;
3 - y2 - 1;
3 - y3 - 1;
4 - y3 - 1;
END;
//Traces completes
optionelles
END;
//Traces obligatoires
STATE 2 : CHOIX : y1;y2;
STATE 3 : CHOIX : y2;
STATE 4 : CHOIX : y3;
END;
```

Figure 9.5 : La syntaxe des fichiers associés aux spécifications

Dans le cas de cet exemple, une solution existe. L'outil affichera un message sous le format décrit par la Figure 9.6.

```
THE SPECIFICATION OF THE GENERIC SOLUTION IS IN FILE :
SUBTYPEGENSOL.txt
THE INPUT ALPHABET IS : {x1, x3, u}
THE OUTPUT ALPHABET IS : {z1, z2, z3, z4, y3}
THE NUMBER OF STATES IS : 5
```

Figure 9.6 : Le message affiché par l'outil

Le fichier contenant la spécification de la solution générique *Sol* est placé dans le répertoire courant. Il a la même syntaxe que le fichier associé à la spécification du système complet. Dans le cas de cet exemple, l'automate associé à la solution générique est illusté dans la Figure 9.7.

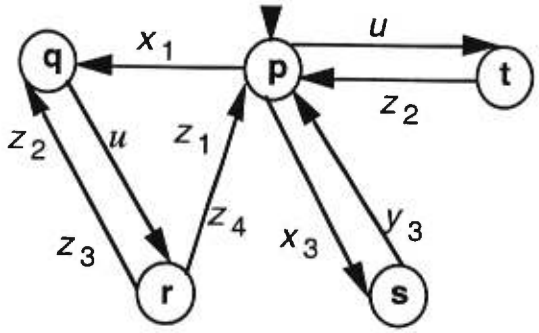


Figure 9.7 : L'automate IOA_{Sol}

Les ensembles de contraintes sur les traces sont :

MT_{Sol} ={(p, {{uz_2}}), (t, Ø), (q, {{u(z_2u + z_3u)*z_1, u(z_2u + z_3u)*z_4}}), (s, {{y_3}}), (r, Ø)}, et

OCT_{Sol} ={(p, {uz_2}), (t, Ø), (q, {u(z_2u + z_3u)*z_1, u(z_2u + z_3u)*z_4}), (s, {y_3}), (r, Ø)}.

9.3. Un exemple illustrant la synthèse d'un contrôleur

Dans ce qui suit nous illustrons à travers un exemple, l'utilisation de l'algorithme 2 du chapitre 7 pour la synthèse d'un contrôleur pour une chaîne de production. L'exemple provient des notes de cours du professeur Wonham [Wonham 94]. Dans cette exemple, la chaîne de production regroupe deux robots R_1 et R_2 ainsi qu'un tapis roulant B. Le robot R_1 (Figure 9.8) peut prendre une pièce à partir d'un ensemble de pièces disponibles. Il doit compléter l'usinage de la pièce puis la déposer sur le tapis roulant. Si pendant l'usinage de la pièce, le robot R_1 tombe en panne, la pièce est perdue et le robot doit être réparé. Les

ensembles des entrées et des sorties de l'automate R_1 sont I_{R_1}={pp, rr1} et O_{R_1}={dpc, r1p}. La signification des actions est :

pp : prendre pièce, rr1 : réparer robot 1,

dpc : déposer pièce complétée, r1p : robot 1 en panne.

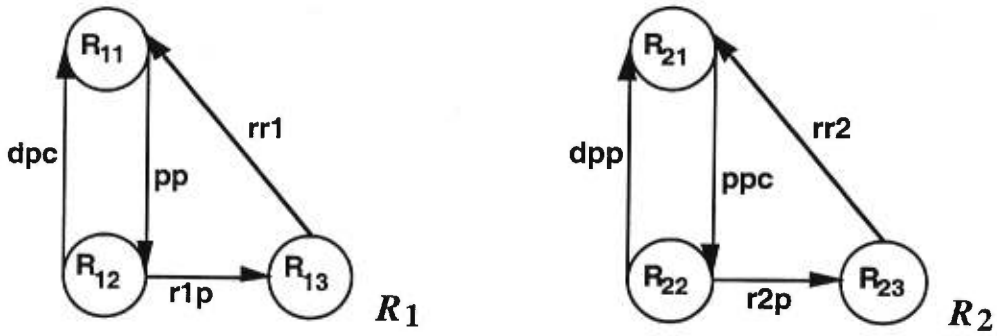


Figure 9.8 : Les automates à entrées et sorties spécifiant le comportement des deux robots

Le robot R_2 (Figure 9.8) peut prendre une pièce dont l'usinage a été complété à partir du tapis roulant. Il doit peindre la pièce puis la déposer sur le tapis roulant. Si pendant la peinture de la pièce le robot R_2 tombe en panne, la pièce est perdue et le robot doit être réparé. Les ensembles des entrées et des sorties de l'automate R_2 sont I_{R_2}={ppc, rr2} et O_{R_2}={dpp, r2p}. La signification des actions est :

ppc : prendre pièce complétée, rr2 : réparer robot 2,

dpp : déposer pièce peinte, r2p : robot 2 en panne.

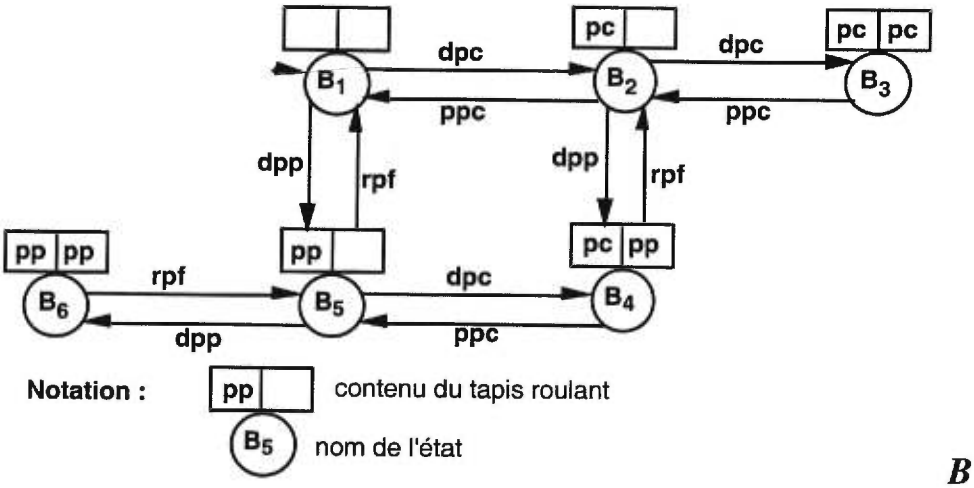


Figure 9.9 : L'automate à entrées et sorties spécifiant le comportement du tapis roulant

Lorsqu'une pièce peinte se trouve sur le tapis roulant (Figure 9.9), ce dernier peut

l'acheminer au dépôt de rangement des pièces. Le tapis roulant à un instant donné peut contenir un maximum de deux pièces. Les ensembles des entrées et des sorties de l'automate B sont I_B={dpc, ppc, dpp} et O_B={rpf} où rpf signifie ranger pièce finie.

La spécification du service désiré pour la chaîne de montage est décrite par l'automate à entrées et sorties avec traces complètes optionnelles $S=(IOA_S, MT_S, OCT_S)$. L'automate à entrées et sorties IOA_S est celui décrit dans la Figure 9.10, il a pour ensemble d'entrées $I_{IOA_S}=\emptyset$ et pour ensemble de sorties O_{IOA_S}={pp, rpf, rr1, rr2}. L'ensemble décrivant les contraintes sur les traces obligatoire est :

$$MT_S=\{(S_1, \{pp\}), (S_2, \{pp, rpf, rr1, rr2\}), (S_3, \{rpf, rr1, rr2\})\}.$$

L'ensemble décrivant les contraintes sur les traces complètes optionnelles est :

$$OCT_S=\{(S_1, \{pp\}), (S_2, \{pp, rpf, rr1, rr2\}), (S_3, \{rpf, rr1, rr2\})\}.$$

les ensembles de contraintes précédants impose que la chaîne de production au départ (dans l'état S_1) doit prendre obligatoirement une pièce. Par la suite (dans l'état S_2), elle peut soit prendre une autre pièce et se retrouver dans l'état S_3, soit acheminer la pièce complétée et peinte vers le dépôt de rangement ou nécessiter la réparation de l'un des deux robots et retourner à l'état S_1. Dans le cas où elle aurait prise une seconde pièce (dans l'état S_3), elle devra acheminer l'une des deux pièces vers le dépôt de rangement ou nécessiter la réparation de l'un des deux robots et retourner à l'état S_2. Un maximum de deux pièces peut se trouver dans la chaîne de production durant son fonctionnement.

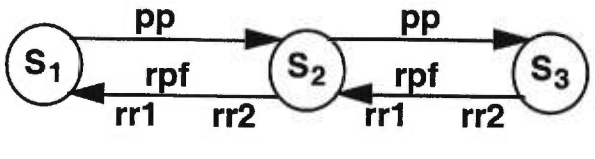


Figure 9.10 : L'automate à entrées et sorties spécifiant le comportement du service désiré

Le comportement libre, c'est à dire non contrôlé, des composantes de la chaîne de production représente le comportement du contexte dans notre approche. Il est décrit dans la table de transition de la Figure 9.11. Pour l'obtention de ce comportement, nous avons d'abord complété les automates à entrées et sorties décrivant le comportement des robots et du tapis roulant, puis nous avons construit leur composition. Cette complétion est nécessaire afin de capturer les comportements où l'une des composantes de la chaîne de production produit une sortie dirigée vers une autre composante qui se trouve dans un état où elle ne peut pas accepter cette action. Par la suite l'outil a été utilisé pour construire le contrôleur requis pour la chaîne de production.

	pp	ppc	rr1	rr2	rpf	dpc	dpp	r1p	r2p
S0	S1	Fail	Fail	Fail	-	-	-	-	-
S1	Fail	Fail	Fail	Fail	-	S2	-	S3	-
S2	S5	S4	Fail	Fail	-	-	-	-	-
S3	Fail	Fail	S0	Fail	-	-	-	-	-
S4	S6	Fail	Fail	Fail	-	-	S7	-	S8
S5	Fail	S6	Fail	Fail	-	S9	-	S10	-
S6	Fail	Fail	Fail	Fail	-	S11	S13	S12	S14
S7	S13	Fail	Fail	Fail	S10	-	-	-	-
S8	S14	Fail	Fail	S0	-	-	-	-	-
S9	S15	S11	Fail	Fail	-	-	-	-	-
S10	Fail	S12	S2	Fail	-	-	-	-	-
S11	S16	Fail	Fail	Fail	-	-	S17	-	S18
S12	Fail	Fail	S4	Fail	-	-	S19	-	S20
S13	Fail	Fail	Fail	Fail	S1	S17	-	S19	-
S14	Fail	Fail	Fail	S1	-	S18	-	S20	-
S15	Fail	S16	Fail	Fail	-	Fail	-	S21	-
S16	Fail	Fail	Fail	Fail	-	S22	S24	S23	S25
S17	S24	S26	Fail	Fail	S2	-	-	-	-
S18	S25	Fail	Fail	S2	-	-	-	-	-
S19	Fail	Fail	S7	Fail	S3	-	-	-	-
S20	Fail	Fail	S8	S3	-	-	-	-	-
S21	Fail	S23	S9	Fail	-	-	-	-	-
S22	S27	Fail	Fail	Fail	-	-	Fail	-	S28
S23	Fail	Fail	S11	Fail	-	-	S29	-	S30
S24	Fail	S31	Fail	Fail	S5	Fail	-	S29	-
S25	Fail	Fail	Fail	S5	-	S28	-	S30	-
S26	S31	Fail	Fail	Fail	S4	-	S32	-	S33
S27	Fail	Fail	Fail	Fail	-	Fail	Fail	S34	S35
S28	S35	Fail	Fail	SS9	-	-	-	-	-
S29	Fail	S36	S17	Fail	S10	-	-	-	-
S30	Fail	Fail	S18	S10	-	-	-	-	-
S31	Fail	Fail	Fail	Fail	S6	S37	S38	S36	S39
S32	S38	Fail	Fail	Fail	S7	-	-	-	-
S33	S39	Fail	Fail	S7	S8	-	-	-	-
S34	Fail	Fail	S22	Fail	-	-	Fail	-	S40
S35	Fail	Fail	Fail	S15	-	Fail	-	S40	-
S36	Fail	Fail	S26	Fail	S12	-	S41	-	S42
S37	S43	Fail	Fail	Fail	S11	-	Fail	-	S44
S38	Fail	Fail	Fail	Fail	S13	Fail	-	S41	-
S39	Fail	Fail	Fail	S13	S14	S44	-	S42	-
S40	Fail	Fail	S28	S21	-	-	-	-	-
S41	Fail	Fail	S32	Fail	S19	-	-	-	-
S42	Fail	Fail	S33	S19	S20	-	-	-	-
S43	Fail	Fail	Fail	Fail	S16	Fail	Fail	S45	S46
S44	S46	Fail	Fail	S17	S18	-	-	-	-
S45	Fail	Fail	S37	Fail	S23	-	Fail	-	S47
S46	Fail	Fail	Fail	S24	S25	Fail	-	S47	-
S47	Fail	Fail	S44	S29	S30	-	-	-	-

Figure 9.11 : La table de transition du contexte

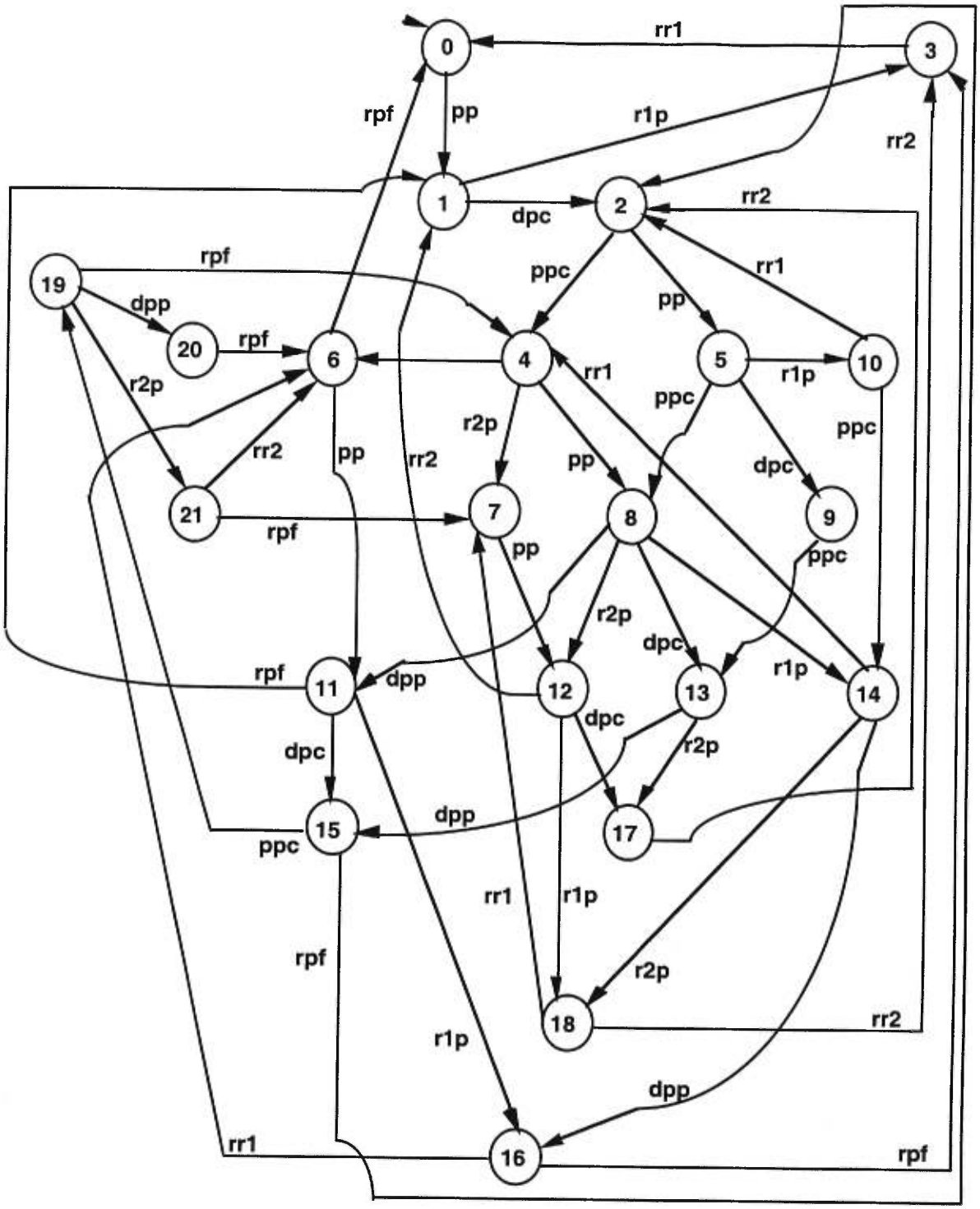


Figure 9.12 : L'automate spécifiant le comportement du contrôleur

Le comportement du contrôleur obtenu est décrit par l'automate à entrées et sorties de

la Figure 9.12. Ce contrôleur est le plus permissif, c'est à dire, qu'il permet tous les comportements acceptables pour les composantes de la chaîne de production.

La chaîne de production de l'exemple précédant est agrandie. On ajoute deux autres robots R_3 et R_4. Le robot R_3 fera le même travail que le robot R_1, alors que le robot R_4 fera le même travail que le robot R_2. De plus, le tapis roulant à un instant donné pourra contenir un maximum de quatre pièces. L'automate à entrées et sorties décrivant le nouveau tapis roulant a 15 états, 6 entrées, une sortie et 70 transitions. Le comportement libre, c'est à dire non contrôlé, des composantes de la chaîne de production est décrit par un automate à entrées et sorties ayant 1179 états, 8 entrées, 9 sorties et 13326 transitions. Le contrôleur pour la chaîne de production obtenu est décrit par un automate à entrées et sorties ayant 351 états, 8 entrées, 9 sorties et 1602 transitions.

	NdC	NEtC	NTC	NEnC	NSC	NEtCt	NTCt	TR (s)
Chaîne1	3	18	98	4	4	12	24	**0,121**
Chaîne2	3	48	276	4	5	22	47	**1,422**
Chaîne3	5	108	865	6	6	48	138	**2,383**
Chaîne4	7	648	6913	8	8	192	712	**18,176**
Chaîne5	5	1179	13326	8	9	351	1602	**74,037**
Chaîne6	9	3888	51841	10	10	768	3473	**851,318**
Chaîne7	7	10719	178364	12	13	1215	6902	**3311,416**

Figure 9.13 : Temps requis par l'outil pour différent exemples

Légende pour le tableau de la Figure 9.13 :

NdC : nombre de composantes dans la Chaîne,
NEtC : nombre d'états du contexte,
NTC : nombre de transitions du contexte,
NEnC : nombre d'entrées du contexte,
NSC : nombre de sorties du contexte,
NEtCt : nombre d'états du contrôleur,
NTCt : nombre de transitions du contrôleur,
TR : temps requis pour la génération du contrôleur en secondes.

Le tableau de la figure 9.13 illustre le temps de calcul (en secondes) nécessaire pour la génération du contrôleur pour des exemples de différentes tailles. Nous avons utilisé pour

cette simulation une machine munie d'un PentiumII 400 MHz. Nous remarquons grâce a ce tableau que le temps de calcul depend certes de la compléxité du système, mais que la compléxité théorique décrite dans le chapitre 7 est loin d'être atteinte.

9.4 Conclusion

Nous avons présenté dans ce chapitre une description rapide de l'outil développé pour la construction de sous-modules. Le langage de programmation Java a été utilisé pour implanter l'outil. Ceci a permis le développement d'un outil facile à utiliser. Il implante les algorithmes pour la construction de sous-modules dans le modèle des automates à entrées et sorties ainsi que le modèle des machines de Mealy. Différentes relations de conformité peuvent être utilisées. Dans le modèle des machines de Mealy, on peut utiliser les relations équivalence, quasi-équivalence et réduction. Dans le modèle des automates à entrées et sorties, on peut utiliser les relations de conformité réalisation sécuritaire et implantation conforme. Cette dernière relation est générique, elle permet de simuler les relations équivalence, quasi-équivalence et réduction selon le choix des contraintes imposées sur les traces de la spécification du système complet désiré.

CHAPITRE 10

CONCLUSION

Nous avons présenté dans cette thèse un travail relié au problème de la construction de sous-modules. Ce genre de problème se rencontre lors de la conception hiérarchique de systèmes complexes, dans la synthèse de contrôleurs, dans la construction de passerelles entre deux réseaux ainsi que dans la réutilisation de composantes. Le problème de construction de sous-modules consiste dans la construction d'un sous-module manquant d'un système lorsque les spécifications du système entier et de tous ses sous-modules à l'exception de celle du sous-module manquant sont données.

Dans notre travail, on s'est concentré principalement sur la résolution du problème de la construction de sous-modules dans le cadre des systèmes communiquant par entrées et sorties, c'est-à-dire, par envoi et réceptions de messages. Le modèle choisi pour décrire les spécifications est celui des automates à entrées et sorties [Lynch 88]. Dans ce modèle, l'ensemble des actions est partionné en deux sous ensembles, les entrées, c'est-à-dire les réceptions, et les sorties, c'est-à-dire les émissions. Cette subdivision permet une distinction claire entre les actions contrôlées par le système - les sorties- et celles qui sont contrôlées par son environnement -les entrées. De plus, les événements observables par un système sont ceux reliés à des actions présentes dans l'alphabet de sa spécification. Afin de modéliser la communication entre deux sous-systèmes, nous avons choisi de faire une composition synchrone d'une entrée avec une sortie, c'est-à-dire qu'une sortie d'un sous système se synchronise avec l'entrée de son correspondant. Dans ce type de communication, une émission d'un message ne peut jamais être refusée. C'est cette absence de refus qui caractérise un modèle de communication par entrées et sortie, à la différence du mode de communication par rendez-vous, comme en LOTOS [IS8807 89], où la notion de refus est au contraire très importante.

Ce modèle généralise celui des machines de Mealy [Gill 62] où une transition est étiquetée par un couple (entrée, sortie). Historiquement, les machines de Mealy ont été conçues comme modèles pour étudier les propriétés de circuits électroniques où les sorties sont synchrones avec les entrées. Cette hypothèse de synchronisme n'est pas réaliste dans le cas de beaucoup de systèmes tels que les réseaux ou les protocoles. De plus, dans ces systèmes on a parfois besoin de représenter des états où un choix est offert entre une entrée et une sortie (la collision de messages dans les protocoles par exemple). Cette situation ne peut être modélisé au moyen de machines de Mealy qu'en admettant que l'interaction vide puisse apparaître aussi bien pour les entrées que pour les sorties.

Le problème de construction de sous-modules est décrit formellement par l'équation $(C||X)\Re A$, où C représente la spécification de la partie connue ou existante du système, appelée le contexte, A représente la spécification du système entier, $||$ décrit le mode de regroupement des module, c'est-à-dire un opérateur de composition, et $\Re$ décrit une relation de conformité entre systèmes. Dans cette formulation, l'ensemble des entrées du module à concevoir est défini de façon implicite. C'est la réunion des sorties internes du contexte et des actions reçues par le système à partir de l'environnement ne faisant pas partie des entrées du contexte. Nous avons jugé plus opportun de laisser le choix de cette ensemble au concepteur. Dans cette optique, nous avons opté pour une définition explicite de cet ensemble afin de permettre au module à concevoir d'observer certaines interactions du contexte avec l'environnement. Dans le cas extrême, le module à concevoir peut observer toutes les action présentes dans le système, c'est le cas dans le travail de [Maler 95]. Par contre, dans les travaux antérieurs utilisant le modèle des machines de Mealy, nous avons le cas de l'observabilité minimale pour le module à concevoir. Le problème de construction de sous-modules consiste dans ce cas à résoudre l'équation $(C||X)\Re A$ sous la contrainte $I_X=In$ où In est l'ensemble d'actions requis pour le module à concevoir.

Au lieu de développer pour chaque relation de conformité un algorithme indépendant pour résoudre le problème de construction de sous-modules, nous avons défini une relation de conformité générique, l'implantation conforme. Cette relation est inspirée de la relation sous-type présente dans les langages orienté-objet. C'est une relation de conformité générique dans le sens où chacune des relations de conformité bien connues, comme trace équivalence, réduction et quasi-équivalence, peut être considérée comme un cas particulier. Pour définir cette relation, nous avons utilisé un nouveau type d'automates que nous avons appelé automates à entrées et sorties avec traces complètes optionnelles. Ces automates permettent de prendre en compte la notion de tâche complète, qui impose qu'après une trace

donnée le système doit progresser afin de produire certaines sorties ainsi que la notion de choix, qui permet à une implantation conforme d'avoir au moins un comportement parmi un ensemble de comportements décrits par la spécification.

Par la suite, nous avons proposé une méthode pour la résolution du problème de construction de sous-modules dans le cadre décrit précédemment. Un algorithme a été développé pour l'obtention de la solution la plus générale lorsque la relation de conformité utilisée est la relation implantation conforme. De plus, nous avons donné une caractérisation de l'ensemble des solutions possibles de l'équation. Par la suite, une implantation de l'algorithme a été réalisée dans le cadre du développement d'un outil pour la construction de sous-modules. Un dernier travail a porté sur l'extension de l'algorithme précédant au cas des systèmes temps-réel.

Nous énumérons dans ce qui suit les principales contributions de cette thèse :

1 - Résolution du problème de construction de sous-modules dans le cas où les spécifications du système désiré et du contexte sont données sous la forme de Machines de Mealy déterministes complètement spécifiées. La relation de conformité utilisée est la relation d'équivalence.

2 - Développement d'une méthode pour la détermination d'une implantation minimale dont les traces sont incluses dans celles d'une spécification donnée sous la forme d'une Machine de Mealy non déterministe complètement spécifiées.

3 - Résolution du problème de construction de sous-modules dans le cas où les spécifications du système désiré et du contexte sont données sous la forme d'automates à entrées et sorties de manière uniforme pour plusieurs relations de conformité et différents cas d'observabilité. Un premier algorithme permet de générer la solution générique de l'équation $(C||X)\Re A$ sous la contrainte $I_X=In$, lorsque la relation de conformité utilisée est la réalisation sécuritaire. Un deuxième algorithme permet de générer la solution générique de l'équation $(C||X)\Re A$ sous la contrainte $I_X=In$, lorsque la relation de conformité utilisée est l'implantation conforme. De plus, dans chacun des cas nous avons caractérisé l'ensemble des solutions possibles de l'équation.

4 - Développement d'un outil progammé en Java pour la construction de sous-modules implantant les différents algorithmes obtenus.

5 - Résolution du problème de construction de sous-modules dans le cas où les spécifications du système désiré et du contexte sont données sous la forme d'automates à entrées et sorties temporisés pour différents cas d'observabilité lorsque la relation de conformité utilisée est la réalisation sécuritaire.

Plusieurs directions pour des travaux futurs peuvent être envisagées. Une première avenue de recherche consisterait à étendre le travail théorique aux systèmes temps-réel. Dans un premier travail on pourrait développer un algorithme pour la résolution du problème de construction de sous-modules lorsque la relation de conformité utilisée serait une adaptation de la relation implantation conforme au cas temporisé. De plus, plusieurs hypothèses ont été requises dans notre travail sur le cas temporisé, il serait intéressant d'essayer d'éliminer ces hypothèses. Par exemple, le cas où des actions non observables par le module à concevoir contiendraient des remises à zéro d'horloges. Un autre travail intéressant serait de compléter l'outil développé en Java pour qu'il prenne en compte le cas des systèmes temps-réel.

BIBLIOGRAPHIE

[Alpern 85] B. Alpern and F. B. Schneider, *Defining liveness*, Information Processing Letters, vol. 21, pp 181-185, 1985.

[Alur 94] R. Alur and D. L. Dill, *A theory of timed automata*, Theoretical Computer Science, 126:183-235, 1994.

[Alur 94a] R. Alur, L. Fix and T. A. Henzinger, *A determinizable class of timed automata*, In Proceedings of CAV'94, Lecture Notes in Computer Science, vol. 818, pp. 1-13, 1994.

[America 85] P. America, *Inheritance and Subtyping in a Parallel Object-Oriented Language*, Philips Research Laboratories, Eindhoven, the Netherlands.

[Aziz 95] A. Aziz, F. Balarin, R. K. Brayton, M. D. DiBenedetto and A. Saldanha, *Supervisory Control of Finite State Machines*, Proceedings of the 7th International Conference, CAV'95, pp279-292, July 3-5, 1995, Liège, Belgium.

[Belina 89] F. Belina and D. Hogrefe, *The CCITT-Specification and Description Language SDL*, Computer Networks and ISDN Systems, Vol. 16, pp.311-341, 1989.

[Bérard 98] B. Bérard, V. Diekert, P. Gastin and A. Petit, *Characterization of the expressive power of silent transitions in timed automata*, In Fundamenta Informaticae, 36(2):145-182, 1998.

[Bérard 96] B. Bérard, P. Gastin and A. Petit, *On the power of non observable actions in timed automata*, In Proceedings of STACS'96, number 1046 in Lecture Notes in Computer Science, pp. 257-268, Springer Verlag, 1996.

[Black 87] A. Black, N. Hutchinson, E. Jul, H. Levy and L. Carter, *Distribution and Abstract Types in Emerald*, IEEE Transaction on Software Engineering, vol. SE-13, no. 1, pp. 65-76, January 1987.

[Bochmann 90] G. v. Bochmann, *Deriving Protocol Converters for Communication Gateways*, IEEE Transaction on Communications, vol. 38, no. 9, Sept. 1990.

[Brinksma 88] E. Brinksma, *A theory for the derivation of tests*, Proc. IFIP Symposium

on Protocol Specification, Testing and Verification, Atl. City, 1988.

[Brinksma 89] E. Brinksma, R. Alderden, R. Langerak, J. v. d. Lagemaat and J. Tretmans, *A formal approach to conformance testing*, Proceedings of the international workshop on Protocol Test Systems, Berlin (West), Germany, October 1989, pp.311-325.

[Chow 78] T. S. Chow, *Testing Software Design Modeled by Finite-State Machines*, IEEE Transactions on Software Engineering, vol. SE-4, NO. 3, May, 1978.

[Courcoubetis 91] C. Courcoubetis and M. Yannakakis, *Minimum and maximum delay problems in real-time systems*, In proceedings of CAV'91, number 575 in Lecture Notes in Computer Science, pages 399-409, Spinger Verlag, 1991.

[Damiani 94] M. Damiani, *Nondeterministic finite-state machines and sequential don't cares*, Proceedings The European Design and Test Conference. EDAC, The European Conference on Design Automation. ETC, European Test Conference. EUROASIC, The European Event in ASIC Design. IEEE Comput. Soc. Press, p. 192-198, 28 Feb.-3 March 1994.

[Daws 96] C. Daws and S. Yovine, *Reducing the number of clock variables of timed automata*, In Proceedings of RTSS'96, Whasington DC, USA, Dec. 4-6, 1996.

[Drissi 99a] J. Drissi and G. v. Bochmann, *Submodule construction for systems of I/O automata*, Technical Report no. 1133, DIRO, University of Montreal, 1999.

[Drissi 99b] J. Drissi and G. v. Bochmann, *Submodule construction tool*, In the proceedings of Int. Conf. on Computational Intelligence for Modelling, Control and Automation (CIMCA'99), vol. 1, pp. 319-324, Vienna 1999.

[Drissi 99c] J. Drissi and G. v. Bochmann, *Submodule construction for systems of timed I/O automata*, submitted for publication to RTSS'2000.

[Drissi 98] J. Drissi, N. Yevtushenko, A. Petrenko and G. v. Bochmann, *On the design of a submodule based on the input/output FSM model*, Technical Report no. 1120, DIRO, University of Montreal, 1998.

[Emerson 88] E. A. Emerson and C. S. Jutla, *The complexity of tree automata and logics of programs*, In 29th Annual Symposium on Foundation of Computer Science, pp 328-337, 1988.

[Fujiwara 91a] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou and A. Ghedamsi,

Test selection based on finite state models, IEEE Transactions on Software Engineering, Vol.17, no.6, June 1991, pp. 591-603.

[Gill 62] A. Gill, *Introduction to the theory of Finite-State Machines*, Mc Graw-Hill Book Company, Inc.

[Glabbeek 90] R. J. v. Glabbeek, *The Linear Time-Branching Time Spectrum*, Proceedings of CONCUR'90, Theories of Consurrency : Unification and Extension, pp. 278-297, Amsterdam, The Netherlands, August 27-30, 1990.

[Grasselli 65] A. Grasselli and F. Luccio, *A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks*, IEEE Transactions on Electronic Computer, pp 350-359, June 1965.

[Green 86] P. E. Green, *Protocol Conversion*, IEEE Transaction on Communications, vol. 34, no. 3, March 1986.

[Haghverdi 96] E. Haghverdi and H. Ural, *An Algorithm for Submodule Construction*, Technical report of the Department of computer Science, University of Ottawa, 1996.

[Halbert 86] D. C. Halbert and P. D. O'Brien, *Using Types and Inheritance in Object-Oriented Languages*, Digital Equipement Corporation, Object-Based Systems Group, HLO 2-3/M08, 77 Reed Road, Hudson, Massachusetts 01749, USA.

[Hoare 85] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.

[Hoffman 93] D. Hoffman and P. Strooper, *A Case Study in Class Testing*, Presented at the IBM Center for Advanced Studies Fall Conference, October 1993, Toronto.

[Hopcroft 87] J. E. Hopcroft and J. D. Ullman, *An Introduction to Automata Theory, Languages, and Computation*, Reading, Mass., Addison-Wesley, 1987.

[IS8807 89] IS8807, *LOTOS: A formal description technique*,

[ISO9074 89] ISO9074, *Estelle: A formal description technique based on an extended state transition model*,

[Kam 95] T. Y. Kam, *State Minimization of Finite State Machines using Implicit Techniques*, Thèse de doctorat, EECS, University of California at Berkley, 1995.

[Kelekar 93] S. G. Kelekar and George W., Hart *Synthesis of protocols and protocol converters using the submodule construction approach*, Center for

Telecommunication Research, Columbia University, New York, # CU/CTR/TR 322-93-01, January 93.

[Kelekar 94] S. G. Kelekar George W. Hart, *Synthesis of protocols and protocol converters using the submodule construction approach*, Proceedings of Protocol Specification, Testing and Verification, XIII, A. Danthine, G. Leduc, P. Wolper (Editors), 1994.

[Kohavi 78] Z. Kohavi, *Switching and Finite Automata Theory*, McGRAW-HILL Computer Science Series.

[Labroue 98] A. Labroue, *Conditions de vivacité dans les automates temporisés*, Research Report LSV-98-7, Sep. 1998, Laboratoire Spécification et Vérification, Ecole normale supérieure de Cachan, France. http://www.lsv.ens-cachan.fr.

[Luo 93] G. Luo, A. Petrenko and G. v. Bochmann, *Selecting test sequences for partially-specified nondeterministic finite state machines*, Technical report #864, University of Montreal, 1993.

[Lynch 88] N. A. Lynch and M. R. Tuttle, *An introduction to input/output automata*, MIT/LCS/TM-373, Laboratory for computer science, Massachusetts Institute of Technology, Nov. 1998.

[Maler 95] O. Maler, A. Pnueli and J. Sifakis, *On the Synthesis of Discrete Controllers for Timed Systems*, In Proceedings of 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, Germany, March 1995, Lecture Notes in Computer Science, vol. 900, pp. 229-242.

[Merlin 83] P. Merlin and G. v. Bochmann, *On the Construction of Submodule Specifications and Communication Protocols*, ACM Trans. on Programming Languages and Systems, Vol. 5, No. 1 (Jan. 1983), pp. 1-25.

[Milner 80] R. Milner, *A calculus of communicating systems*, Lecture Notes in Computer Science, No. 92, Springer Verlag, 1980.

[Negulescu 95] R. Negulescu and J. A. Brzozowski, *Relative Liveness : from intuition to automated verification*, Research report CS-95-32, University of Waterloo, Canada, 1995.

[Park 81] D. Park, *Concurrency and automata on infinite sequences*, Proceedings of 5th GI Conference, LNCS 104, 1981.

[Parrow 89] J. Parrow, *Submodule Construction as Equation Solving in CCS*,

Theoretical Computer Science, 68, pp 175-202, 1989.

[Petrenko 98] A. Petrenko and N. Yevtushenko, *Solving asynchronous equations*, in proceedings of FORTE'98, France, November 98.

[Petrenko 93] A. Petrenko, N. Yevtushenko, A. Lebedev and A. Das, *Nondeterministic state machines in protocol conformance testing*, Proceedings of the IFIP Sixth International Workshop on Protocol Test Systems, Pau, France, September 1993, pp. 363-378.

[Pfleeger 73] C. P. Pfleeger, *State reduction in incompletly specified finite state machines*, IEEE Transactions on Computers, pp. 1099-1102, October 1973.

[Phalippou 94] M. Phalippou, *Relations d'implantation et hypothèses de test sur des automates à entrées et sorties*, Thèse de Doctorat, Bordeaux, France, 1994.

[Pyne 61] I. B. Pyne and E. J. McCluskey, *An essay on prime implicant tables*, Journal SIAM, vol. 9, pp. 604-631, January 1961.

[Qin 91] H. Qin and P. Lewis, *Factorisation of finite State Machines under Strong and Observational Equivalences*, Journal of Formal Aspects of Computing, Vol. 3, pp 284-307, July-Sept. 1991.

[Ramadge 87] P. J. Ramadge and W. M. Wonham, *Supervisory Control of a Class of Discrete Event Processes*, SIAM Journal of Control and Optimization 25, pp 206-230, 1987.

[Ramadge 89] P. J. Ramadge and W. M. Wonham, *The Control of Discrete Event Systems*, Proceedings of the IEEE 77(1), pp 81-98, 1987.

[Shields 89] M. W. Shields, *Implicit System Specification and the Interface Equation*, Computer Journal, Vol. 32, 5, pp. 399-412, Oct. 1989.

[Sidhu 89] D. P. Sidhu and T. K. Leung, *Formal methods for protocol testing: a detailed study*, IEEE Tr. on SE, Vol 15, 4 (1989).

[Somé 97] S. Somé, *Dérivation de specification à partir de Scénarios d'interaction*, Thèse de doctorat, DIRO, Université de Montréal, 1997.

[Starke 72] P. H. Starke, *Abstract automata*, American Elsevier Publishing Company, Inc-New York.

[Tan 95] Q. M. Tan, A. Petrenko, G. v. Bochmann and G. Luo, *A framework for conformance testing of systems communicating through rendezvous*, in proceedings of the 26th IEEE International Symposium on Fault-Tolerant Computing, pp. 230-238, Sendai, Japan, June 25-27, 1996.

[Tao 95] Z. P. Tao, G. v. Bochmann and R. Dssouli, *An Efficient Method for Protocol Conversion*, Proceeding of the Fourth International Conference on Computer Communications and Networks, Sept. 20-23, 1995, Las Vegas, Nevada, USA.

[Tao 96] Z. Tao, *Formal Method for the Design of Real-Time Communicating Subsystems and Controllers*, Thèse de doctorat, DIRO, Université de Montréal, 1996.

[Tarski 55] A. Tarski, *A Lattice-Theoretical Fixpoint Theorem and its applications*, Pacific Journal of Math. 5, 1955.

[Thistle 94] J. G. Thistle and W. M. Wonham, *Control of infinite behaviour of finite automata*, SIAM Journal of Control and Optimization 32(4), pp 1075-1097, 1994.

[Thistle 95] J. G. Thistle and W. M. Wonham, *On Control of Systems Modelled as Deterministic Rabin Automata*, Discret Event Dynamic Systems : Theory and Applications, 5, pp 357-381, 1995.

[Turner 92] C. D. Turner and D. J. Robson, *The Testing of Object-Oriented Programs*, Technical Report TR-13/92, Computer Science Division, School of Engineering and Computer Sciences (SECS), University of Durham, England.

[Unger 69] S. H. Unger, *Asynchronous Sequential Switching Circuits*, Wiley-Interscience, New York, 1969.

[Ural 87] H. Ural, *A test derivation method for protocol conformance testing*, in Proc. IFIP Symp. on Protocol Specification, Testing and Verification VII, North Holland Publ., 1988, pp. 347-358.

[Watanabe 93a] Y. Watanabe and R. K. Brayton, *The Maximum Set of Permissible Behaviors for FSM Networks*, Proc. of the IEEE/ACM International Conference on Computer-Aided Design (1993) pp 316-320.

[Watanabe 93b] Y. Watanabe and R. K. Brayton, *State Minimization of Pseudo Non-Deterministic FSM's*, Proc. of the IEEE/ACM International Conference on Computer-Aided Design (1993) pp 184-191.

[Wonham 94] W. M. Wonham, *Notes on Control of Discrete-Event Systems*, System Control Group ECE1636F/1637S Course Notes. Department of Electrical Engineering, University of Toronto, Toronto.

[Wood 87] D. Wood, *Theory of Computation*, John Wiley & Sons, Inc, 1987.

[Yao 93] M. Yao, A. Petrenko and G. v. Bochmann, *Conformance testing of*

protocol machines without reset, IFIP 13th Int. Conference on Protocol Specification, Testing and Verfication, pp.241-253, Liege, Belgium, May 1993.